

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 06-068053

(43)Date of publication of application : 11.03.1994

(51)Int.Cl.

G06F 15/16

(21)Application number : 04-221592

(71)Applicant : TOSHIBA CORP

(22)Date of filing : 20.08.1992

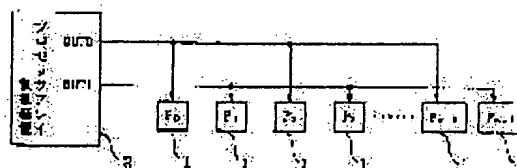
(72)Inventor : FUJII HIROSHIGE
YOSHIDA TAKASHI
SATO HISATOMO
TAKAHASHI MASASHI

(54) PARALLEL COMPUTER

(57)Abstract:

PURPOSE: To shorten the execution time and to effectively use processors even in the case of the degree of parallelism lower than the number of processors.

CONSTITUTION: A processor array consisting of n processors 1 and a processor array controller 2 which controls the processor array and extracts two kinds of instruction strings, which can be executed in parallel, from instruction strings to supply them to the processor array are provided, and one of two kinds of instruction strings supplied from output terminals OUT0 and OUT1 of the processor array controller 2 is received by each processor 1, and two kinds of instruction strings are executed in parallel by n processors.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁(JP)

(12) 公開特許公報(A)

(11)特許出願公開番号

特開平6-68053

(43)公開日 平成6年(1994)3月11日

(51)Int.Cl.⁵

G 0 6 F 15/16

識別記号

3 9 0 T 9190-5L

庁内整理番号

F I

技術表示箇所

審査請求 未請求 請求項の数5(全27頁)

(21)出願番号 特願平4-221592

(22)出願日 平成4年(1992)8月20日

(71)出願人 000003078

株式会社東芝

神奈川県川崎市幸区堀川町72番地

(72)発明者 藤井 洋重

神奈川県川崎市幸区小向東芝町1 株式会社東芝総合研究所内

(72)発明者 吉田 尊

神奈川県川崎市幸区小向東芝町1 株式会社東芝総合研究所内

(72)発明者 佐藤 寿倫

神奈川県川崎市幸区小向東芝町1 株式会社東芝総合研究所内

(74)代理人 弁理士 三好 秀和 (外1名)

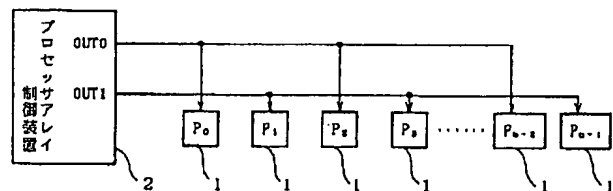
最終頁に続く

(54)【発明の名称】 並列計算機

(57)【要約】

【構成】 n個のプロセッサ1からなるプロセッサアレイと、このプロセッサアレイを制御すると共に、命令列から並列実行可能な2種類の命令列を抽出してプロセッサアレイに供給するプロセッサアレイ制御装置2とを備え、プロセッサアレイ制御装置2の出力端子OUT0、1から供給される2種類の命令列のうちの1種類を各プロセッサ1が受け取り、2種類の命令列をn個のプロセッサ1で並列実行する。

【効果】 並列度がプロセッサ数より小さい場合でも、実行時間の短縮とプロセッサの有効利用を図れる。



【特許請求の範囲】

【請求項 1】 複数のプロセッサからなるプロセッサアレイと、

このプロセッサアレイを制御すると共に、並列実行可能な複数種類の命令列を前記プロセッサアレイに供給するプロセッサアレイ制御手段とを備え、

前記プロセッサアレイ制御手段から供給される複数種類の命令列の中の 1 種類ずつを各プロセッサが受け取り、前記複数種類の命令列を前記複数のプロセッサで並列実行することを特徴とする並列計算機。

【請求項 2】 単一の制御プロセッサと、

該制御プロセッサからアクセス可能な第 1 のメモリと、制御プロセッサから転送される命令情報にしたがって、処理を行う複数の演算プロセッサと、該演算プロセッサからアクセス可能な第 2 のメモリとから構成され、

さらに前記演算プロセッサには、演算プロセッサでの実行を制御する命令情報の格納手段と、

該格納手段に、前記制御プロセッサからの命令情報を入力するか、第 2 のメモリに格納されている命令情報を格納するかを選択する手段と、

前記格納手段に格納されている命令情報にしたがって演算を実行する手段とを備えることを特徴とする並列計算機。

【請求項 3】 個々の通信経路で接続されている複数の演算要素と、

解く問題のサイズに応じて演算に使用する演算要素と使用しない演算要素とを識別する識別手段と、

この識別手段によって識別された、演算に使用しない演算要素をバイパスするバイパス手段とを備えることを特徴とする並列計算機。

【請求項 4】 個々の通信経路で接続されている複数の演算要素と、

各々の演算要素と個々に結合しているローカルメモリと、

解く問題のサイズに応じて演算に使用する演算要素と使用しない演算要素とを識別する識別手段と、

この識別手段によって識別された、演算に使用しない演算要素と結合しているローカルメモリを、演算に使用する演算要素から直接アクセス可能にするアクセス手段とを備えることを特徴とする並列計算機。

【請求項 5】 複数のプロセッサが N 次元ネットワークで相互接続され、

各プロセッサが、

隣接するプロセッサからメッセージを受信し、出力先を決定する出力先決定手段と、

出力先ごとに設けられ、前記出力先決定手段によって出力先が決定されたメッセージを受信した順に蓄える蓄積手段と、

蓄積されたメッセージを、決定された出力先のプロセッサに出力する出力手段とを備えたことを特徴とする並列計算機。

【発明の詳細な説明】

【0001】

【産業上の利用分野】 本発明は、並列計算機に関する。

【0002】

【従来の技術】 大規模な科学技術計算を高速に実行するために、並列計算機が期待されている。並列計算機の 1 つとして SIMD 型並列計算機がある。SIMD 型並列計算機は、複数のプロセッサ上に配置された複数のデータに対して、すべてのプロセッサで同じ命令を実行する。この場合、プロセッサ数だけのデータに対する処理を並列に実行することになる。

【0003】 一般に、並列計算機上で実行しようとするプログラムには、並列度というものがあり、それ以上のプロセッサがあっても、速度は向上しない。例えば、繰り返し処理を並列実行するときに、1 回の繰り返しを 1 台のプロセッサで実行することを考えると、1000 回の繰り返し処理を 10000 台のプロセッサで実行しても、1000 倍以上には速度向上は望めない。したがって、プロセッサの数よりも少ない並列度の問題を解く場合には、実質的に使用されないプロセッサが多くなり、無駄である。

【0004】 問題の並列度が少ない場合に、実行時間の短縮とプロセッサの有効利用を考える必要がある。例えば、上述の例では繰り返し処理が複数命令からなり、その複数命令の中で並列に実行できる命令があれば、その命令を複数のプロセッサで並列実行できる可能性がある。しかしながら、このような複数命令の並列実行は、単一命令を全プロセッサが実行するという SIMD 型並列計算機では、実現するのは困難である。

【0005】 ここで、一般的な並列計算機について図 6 を用いて説明する。

【0006】 同図で、この並列計算機は、大きく分けて、1 台の制御プロセッサ (CP) 11 と、複数台の演算プロセッサ (PE) 12 から構成される。制御プロセッサ 11 には全体メモリ (M) 13 が接続され、演算プロセッサ 12 にはそれぞれローカルメモリ (LM) 14 が接続される。制御プロセッサ 11 と各演算プロセッサ 12 は、グローバルバス 15 で接続される。演算プロセッサ 12 は、ネットワーク 16 で相互接続される。ネットワーク 16 の形態としては、格子結合、2 進 n キューブ結合、木結合等、多くのものが知られている。

【0007】 制御プロセッサ 11 は、それ自体が一つの情報処理装置であり、全体メモリ 13 に格納されているプログラム (機械語命令列) を読み込み、解読する。解読した命令が制御プロセッサ 11 で実行すべき命令であれば制御プロセッサ 11 自身で実行する。解読した命令が演算プロセッサ 12 で実行すべき命令であれば、制御

プロセッサ11はこの命令をグローバルバス15を介して全演算プロセッサ12に放送し、演算プロセッサ12でこの命令の実行が行われる。

【0008】制御プロセッサ11は、グローバルバス15を介して、すべての、あるいは特定の演算プロセッサ12から、データの読み出し、および書き込みを行うことができる。また、同期バス17を介して、すべての演算プロセッサ12の状態の論理和を認識することができる。

【0009】さらに、制御プロセッサ11は、フロントエンド計算機18と結合されている。フロントエンド計算機18では、制御プロセッサ11および演算プロセッサ12で実行するプログラムの開発を行ったり、処理すべきデータの入出力を行う。

【0010】演算プロセッサ12は、制御プロセッサ11から放送される命令に従って、自ローカルメモリ14のデータに対して演算を行ったり、ネットワーク16を介して他の演算プロセッサ12とデータ通信を行ったりして処理を行う。

【0011】演算プロセッサ12は、例えば、図37に示すような構成になっている。

【0012】命令レジスタ24には、制御プロセッサ11から放送される命令が、1クロックごとに格納される。演算プロセッサ12を構成する他のすべての要素、すなわち、アドレスレジスタ(AR)26、データレジスタ(DR)27、状態制御レジスタ(PSW)31、プロセッサ番号レジスタ(PNR)32、ALU29、レジスタファイル30、通信制御部33は、命令レジスタ24の内容によって制御される。

【0013】制御プロセッサ11からは、一時期には常に同じ命令が全演算プロセッサ12に対して放送され、全演算プロセッサ12が一斉に同じ処理を行うことになる。いわゆるSIMD(Single Instruction Multiple Data Stream)方式と呼ばれるものである。一方、各演算プロセッサ12が独自にプログラムを持ち、各演算プロセッサ12が独立して動作する方式も提案されており、MIMD(Multiple Instruction Multiple Data Stream)方式と呼ばれる。

【0014】図37に示したSIMD方式は、MIMD方式に比較して制御が簡単でハードウェアが少なく済み、多くのデータに対して同じ処理を行う必要のある場合に有効である。しかし、データによって異なった処理が必要となる場合に、効率よく処理できないという欠点があり、この場合はMIMD方式が有利になる。

【0015】このような、並列計算機においては、各演算プロセッサ(演算要素/以下、PE: Processing Elementという)間の接続方式がその並列計算機をどのようにするかがその並列計算機の特徴を出す大きな要因となっている。PE間接続方式は大きく分けて、静的結合と動的結合の2つがある。

【0016】静的結合は、隣接する通信相手のPEが経路によって固定されているもので、リング結合、トラス結合、木構造結合などがある。その特徴は、通信経路が確定しているため、隣接経路のあるPE間でのデータ通信は速いが、逆に遠くのPE、すなわち他のPEを経由して通信を行わなければならない相手との通信は非常に遅くなるという問題がある。

【0017】動的結合は、通信相手が固定でなく、クロスバスイッチを多段に設けて通信時にスイッチを切り替え、通信相手を決定するもので、クロスバネット、オメガネットなどがある。その特徴は、スイッチ群を経由することにより、ほぼ全てのPEへの通信距離が等価に見えることであり、任意のPE間の通信時間がほぼ一定となることである。

【0018】また、PEが故障した場合の故障PEの切り放しも容易に行うことが出来る。しかし、経路の確定のためと、スイッチのディレイのために、通信のための時間がかかり、また、スイッチでのデータの衝突などが問題となる。

【0019】これらの結合路の使い分けとして、隣接作用問題でよく行われる隣のPEへのデータ転送では静的結合である隣接結合路を使用し、任意のPEから他の任意のPEへデータを転送するいわゆるランダム通信では動的結合の通信経路を使用する場合が多い。

【0020】静的結合網の並列計算機に、サイズの合う問題を実行させる場合は効率はよいが、サイズの異なる問題を実行させる場合は、PE台数が中途半端になり、非常に効率が落ちてしまう。例えば 32×32 の2次元トラス結合の並列計算機に問題サイズ $32 \times 32 \times 10$ の隣接作用問題を乗せる場合、x軸、y軸をそれぞれPEアレイの各辺に対応させれば効率は非常によい。

【0021】ところが、問題サイズ $30 \times 30 \times 10$ の問題では、x、y軸をPEアレイの各辺に対応させてもそれぞれ2行分のPE列が余ってしまう。この場合は、各2行分が余ったことをプログラマーあるいはコンパイラが認識してこれに見合ったコードを出力しなければならない。

【0022】例えばリング状に隣接PEに接続されている場合、問題サイズのために使用しないPEがあれば、同PEを通過させるために、余分な転送動作を行う必要があり、各々のPEでは使用するPEと使用しないPEの判別動作も必要となってくる。そのため問題サイズとアレイサイズが一致した場合に比べ、データ転送でのオーバーヘッドにより明らかに処理効率が落ちる。

【0023】並列計算機の他の特徴としては、各PEの演算データを保持するメモリの構成も挙げられる。大きく分けると、共有メモリ型と、分散メモリ型がある。共有メモリ型は、各PEとメモリ間を共有バスあるいは動的結合路で結合する場合が多く、比較的プロセッサ台数が少ない場合に有効である。

【0024】一方、分散メモリ型は各々のPEにローカルメモリを付け、データの共有性を無くするかわりにデータアクセスのコンフリクトを解消している。そのため、近年のPE台数の多い並列計算機では、ほぼこの分散メモリ型で構成されている。

【0025】分散方式のローカルメモリでは、解く問題の並列度は低い。1台のPEで分担するデータ量が多い場合、一度に乗せることのできるデータ量は1台のPEのローカルメモリに依存してしまう。稼動しているPEよりもIDLEのPEの方が多い場合、本来1台のPEで受け持つデータを2台のPEに振り分け、2台のPEの連動により、疑似的に1台のPEとして処理を行う。この場合、2台の連動を処理するためのソフトウェアのオーバーヘッドと、隣接通信が行えないオーバーヘッドから、処理能力が低下する。

【0026】以上のように、従来の並列計算機では問題サイズが異なることによるオーバーヘッドが大きいため、以前よりオーバーヘッドを小さくするような並列計算機が必要とされていた。

【0027】一方、複数の演算プロセッサを相互結合する通信ネットワークは、これらの演算プロセッサを相互に結合するが、プロセッサの数は非常に多いので、任意の演算プロセッサから他の全ての演算プロセッサに結合するための通信経路を備えるのは、物理的にも経済的にも現実的ではない。

【0028】そこで、例えば図19に示すように、相互に隣接する演算プロセッサ55間の通信チャンネル56だけを備えさせ、隣接せず直接に結合されていない演算プロセッサ55間では、一つ以上の中継プロセッサを介して通信を行う方法がある。

【0029】このようなプロセッサ間通信方式としては、例えばW. J. Dally 他 の "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks"

(IEEE Trans. Comput., vol. C-6, No. 5, May 1987) に開示されているワームホール・ルーティングがある。このワームホール・ルーティングでは、メッセージをフリットと呼ばれる小さな単位に分割して通信を行う。通信は送信側プロセッサから受信側プロセッサまでの経路を確定して開始され、その経路は通信が終了するまでそのメッセージに占有される。

【0030】ワームホール・ルーティングは以上のような通信を行うので、あるメッセージの通信が行われている間は、その通信で使用されている通信チャンネルを他のメッセージの通信に使用することはできない。したがって、あるメッセージを構成する全てのフリットの通信が途中で停止してしまうと、その通信に使用されている通信チャンネルを必要とする通信は、その間ずっと待ち続けなければならない。すなわち、あるメッセージの通信が他のメッセージの通信を妨害してしまうという問題がある。

【0031】この問題を解決するために、例えばW. J. Dally の "Virtual-Channel Flow Control" (IEEE Trans. Para. and Dist. Sys., vol. 3, No. 2, Mar. 1992) に開示されているように、各演算プロセッサのメッセージの受信側に複数のフリットを蓄えることのできるバッファを設ける。

【0032】こうすることで、通信が停止しているメッセージを演算プロセッサ内に蓄え、これがネットワークに広がる範囲を狭め、このメッセージがさらに他のメッセージの通信を妨害しないようにしてる。

【0033】このような演算プロセッサの従来例を図38に示す。ルータ部51と演算処理部52から構成される。ルータ部51はフリットを蓄えるバッファ53x, 53y, 53pと、フリットの出力先を決定するスイッチ54から構成される。

【0034】Pin, Poutはルータ部51と演算処理部52とを接続する通信チャンネルである。Xin, Xout, Yin, Youtはそれぞれ隣接する演算プロセッサとの間の入力および出力のための通信チャンネルである。

【0035】しかしこのプロセッサを用いた場合、以下のような問題がある。図39において、55a~55eは演算プロセッサであり、それぞれバッファ53x, 53y, 53p、スイッチ54および演算処理部52から構成される。

【0036】出力先の演算プロセッサ55bのバッファが一杯で、通信が停止しているフリット67が演算プロセッサ55aのバッファ53xに蓄えられているとする。このとき演算プロセッサ55cから演算プロセッサ55aに、別のメッセージのフリット68が入力されるとする。このメッセージの出力先が演算プロセッサ55eであるとする、演算プロセッサ55eのバッファに空きがあれば、フリット68は出力されるべきである。

【0037】ところが、バッファ53xに蓄えられているフリット67の通信が停止しているため、フリット68を出力することができない。すなわち、出力先の通信チャンネルが空いており本来出力されるべきであるメッセージが、通信の停止している他のメッセージによって通信を妨害される。したがって、並列計算機全体としての通信時間は増大し、スループットも低下する。

【0038】ところで、図19で示した通信方法とは別に、図29に示すような、通信方法がある。この方法におけるワームホール・ルーティングでは、メッセージをフリットと呼ばれる小さな単位に分割し、中継プロセッサはメッセージが全て到達するのを待たないで、あるフリットが到着したら直ちにそのフリットを次のプロセッサに転送する。

【0039】ワームホール・ルーティングを行うプロセッサの構成例を図40に示す。90はルータ部で、Xin, Xoutはそれぞれ隣接するプロセッサとの間の入

力および出力チャネルである。Pin、Poutはルータ部90と演算処理部72を接続する通信チャネルである。

【0040】図29においてプロセッサ74aからプロセッサ74eへの通信を考える。ワームホール・ルーティングではハンドシェイクを行って通信を行う必要がある。ハンドシェイクのリクエストとアクノリッジとで一回のフリットの通信が成立するので、一単位時間は2と考えることができる。Dをプロセッサ間の距離、Lをフリットの数として、図41のD=4、L=4の場合を例にして、必要な通信時間を考察する。

【0041】例えばプロセッサ74bからプロセッサ74cへ先頭フリットを送信している間に、プロセッサ74aはプロセッサ74bに二番目のフリットを送信することができる。同様の考察から、プロセッサ74aからプロセッサ74eまでの通信を考える。

【0042】先頭フリットがプロセッサ74aからプロセッサ74eに到達するまでに必要な時間は $2 \times D$ であり、この間に二番目のフリットは、プロセッサ74dに到達している。したがって、二番目以降の全てのフリットが、プロセッサ74aから74eに到達するまでに必要な時間は $2 \times (L-1)$ となる。この結果、ワームホール・ルーティングの場合に必要なプロセッサ間の通信時間は $2 \times (D+L-1)$ となる。

【0043】プロセッサ間通信の他の方式には、特開平2-228762に開示されているワームホール・ルーティングを変形した方法がある。この方法ではメッセージとともに伝達されるクロックの制御のもとに、プロセッサ間で通信が行われる。ワームホールでは通信を行うごとに、ハンドシェイクを行って隣接プロセッサ間で同期をとらなければならないのに対して、この方法ではいったんプロセッサ間の経路が決定してしまえば、クロック同期にしたがって通信すればよく、ハンドシェイクを行う必要がない。

【0044】この方法ではいったん経路が確定した後では、ハンドシェイクを行う必要がないので $(D+L-1)$ で通信が行える。しかし、経路を確定するとき、経路の確定を送信側プロセッサへ報告するときには、ワームホール・ルーティングと同様にハンドシェイクを行わなければならない、それぞれ $(2 \times D)$ の通信時間を必要とする。したがってワームホール・ルーティングの変形を用いた場合に必要のプロセッサ間の通信時間は $(5 \times D+L-1)$ となる。図42にD=4、L=4の場合のタイミング図を示す。

【0045】プロセッサの数が1,000を越えるような大規模なネットワークになると、これらのDの項による影響が無視できなくなる。このことはDが大きくなるとそのメッセージの通信に要する時間が増大するだけではない。すなわち、通信中のメッセージがプロセッサやプロセッサ間のチャネルを占有し、他のメッセージはそ

のプロセッサやチャネルを利用することができなくなる。

【0046】そのためプロセッサやチャネルが解放されるのを待たなければならない。つまり、あるメッセージの通信が他のメッセージの通信の妨げとなるわけである。したがってメッセージの通信に時間がかかると、ほかのメッセージの障害となる時間も増加し、ネットワーク全体での通信時間はさらに増大することになる。

【0047】

【発明が解決しようとする課題】以上で述べたように、従来のSIMD型並列計算機では、問題の並列度がプロセッサよりも大きい場合には、その並列度を生かして高速に問題を解くことができるが、問題の並列度がプロセッサ数よりも小さい場合には、有効に利用できるのは問題の並列度分のプロセッサのみで、それ以外のプロセッサは遊ぶことになり、問題を解く時間を短縮するのには役に立たないという問題がある。

【0048】また、SIMD型並列計算機は、多くのデータに対して同じ処理を行う必要がある場合に有効である。しかし、データによって、異なった処理が必要となる場合に、効率よく処理できないという欠点があった。一方MIMD型並列計算機では、データによって異なる処理が必要な場合も柔軟に対応できるが、制御が複雑で、ハードウェアが増大するという欠点があった。

【0049】さらに、従来の静的結合網の並列計算機では、問題のサイズが異なる場合にPE台数が中途半端になり、効率が低下してしまったり、データ量が多い場合に複数台数のPEを連動させるためのオーバーヘッドが大きくなるという問題があった。

【0050】一方、従来の並列計算機では、出力先の通信チャネルが空いており、通信が可能であるにも関わらず、通信が停止しているメッセージによって他のメッセージの通信が妨害されてしまうという問題があった。

【0051】また、ネットワークの規模がプロセッサ間通信に与える影響が大きく、大規模なネットワークでは通信に非常に時間がかかってしまうという問題もあった。

【0052】この問題を解決するために、第1の発明は、問題の並列度がプロセッサ数よりも小さい場合でも、できるだけ多くのプロセッサを効率よく利用できる並列計算機を提供することを目的とする。

【0053】第2の発明は、SIMD型並列計算機に若干のハードウェアを付加するだけで、MIMD処理を可能とする並列計算機を提供することを目的とする。

【0054】第3の発明は、問題のサイズが異なる場合に、使用されないPEをバイパスすることによって処理効率の低下を防ぎ、さらにデータ量が多い場合に、使用されないPEのローカルメモリを使用されるPEからアクセス可能にすることによってオーバーヘッドを小さくすることができる並列計算機を提供することを目的とす

る。

【0055】また、第4の発明は、通信可能なメッセージが妨害されずに、通信時間の短縮を可能とする並列計算機を提供することを目的とする。

【0056】第5の発明は、通信チャネルおよび制御信号を中継プロセッサでバイパスすることにより、通信時間の短縮を可能とする並列計算機を提供することを目的とする。

【0057】

【課題を解決するための手段】上記目的を達成するため、第1の発明は、複数のプロセッサからなるプロセッサアレイと、そのプロセッサアレイに対して並列実行可能な複数種類の命令列を抽出して供給するなどの制御を行なうプロセッサアレイ制御手段から構成される。

【0058】各プロセッサは、プロセッサアレイ制御手段から供給される複数種類の命令列の中の1種類ずつを受け取る機能と、受け取った命令列を他のプロセッサと並列に実行する機能とを持つ。

【0059】また、第2の発明の並列計算機は、大きく分けて、複数の演算プロセッサと、これを制御する単一の制御プロセッサとから構成される。さらに、演算プロセッサは、演算プロセッサでの実行を制御する命令を格納するためのレジスタと、このレジスタに、制御プロセッサからの命令を入力するか、あらかじめメモリに格納されている命令を格納するかを選択するためのマルチプレクサと、レジスタに格納されている命令にしたがって演算を実行する演算部とから構成される。

【0060】第3の発明は、個々の通信経路で接続されている複数の演算要素と、各々の演算要素と個々に結合しているローカルメモリと、解く問題のサイズに応じて演算に使用する演算要素と使用しない演算要素とを識別する識別手段と、この識別手段によって識別された、演算に使用しない演算要素をバイパスするバイパス手段と、前記識別手段によって識別された、演算に使用しない演算要素と結合しているローカルメモリを、演算に使用する演算要素から直接アクセス可能にするアクセス手段とから構成されている。

【0061】さらに、第3の発明は、前記複数の演算要素、識別手段、バイパス手段、ローカルメモリ、及びアクセス手段を1つのチップ上に配置し、前記バイパス手段及びアクセス手段を前記演算要素上に配線する。

【0062】また、第4の発明は、複数のプロセッサがN次元ネットワークで相互接続され、各プロセッサが、隣接するプロセッサからメッセージを受信し、出力先を決定する出力先決定手段と、出力先ごとに設けられ、前記出力先決定手段によって出力先が決定されたメッセージを受信した順に蓄える蓄積手段と、蓄積されたメッセージを、決定された出力先のプロセッサに出力する出力手段とから構成される。

【0063】第5の発明は、複数のプロセッサがN次元

ネットワークで相互接続され、各プロセッサが、隣接するプロセッサからメッセージおよび制御信号を受信し、メッセージの宛先を判定する判定手段と、判定されたメッセージが自プロセッサ以外の場合、受信したメッセージおよび制御信号を隣接するプロセッサにバイパスするバイパス手段とから構成されている。

【0064】

【作用】上記手段により、第1の発明は、対象とする問題の並列度が十分大きい場合には、従来のSIMD型並列計算機と同様の動作をさせる。その場合には、プロセッサアレイ制御手段が、プロセッサアレイへ供給する複数の命令列をすべて同一の命令列にすることによって実現できる。各プロセッサは、プロセッサアレイ制御手段が供給する複数の命令列の一つを受け取り実行することができるから、複数の命令列がすべて常に同一であれば、すべてのプロセッサが常に同一命令を並列実行することになる。

【0065】対象とする問題の並列度がプロセッサ数より小さい場合には、以下のように制御する。ここで、プロセッサアレイ制御手段は、並列実行可能なm種類の命令列を抽出してプロセッサアレイへ供給できるようになっており、プロセッサはn個あるものとする。そして、n個のプロセッサをm個ずつのk組に分ける（すなわち、 $n = m \times k$ 、n、m、kは整数とする）。このm個のプロセッサからなる組をクラスタと呼ぶことにする。

【0066】各クラスタにはすべて同じm種類の命令列を供給し、同じ処理を行なうように制御する。プロセッサアレイ制御手段から供給されるm種類の命令列は、各クラスタ内のm個のプロセッサのそれぞれに1種類ずつ供給され、各プロセッサは、供給された命令列を実行する。

【0067】また、第2の発明は、通常のSIMD処理では、各演算プロセッサは、制御プロセッサから放送される命令にしたがって処理を行う。制御プロセッサから指示があると、あらかじめ各演算プロセッサのローカルメモリに格納されている命令が命令レジスタに格納されるようにマルチプレクサが切り換わる。各演算プロセッサのローカルメモリに異なる命令を格納しておくことによって、MIMD処理を行う。

【0068】さらに、第3の発明は、PEアレイのサイズと問題サイズが異なる場合、コンパイラあるいは他の手段により、使用するPEと、使用しないPEを予め決定し、決定されたPE情報を識別手段に送る。識別手段では、PE毎の使用する／しないの情報を保持しておく。

【0069】処理実行時に隣接通信を行う際、各PEのバイパス手段は自PEに対応する識別手段から、バイパスを行うかどうか指示される。稼働中のあるPEは、隣のPEが休止である場合、休止のPEのバイパス手段により、さらに先の稼働中のPEがあたかも隣のPEに

見える。

【0070】以上から、処理する問題のサイズに合わせて使用するPEと使用しないPEを適切に選択することにより、解く問題に最適なハードウェアのレイサイズを用意する。

【0071】また、アクセス手段に予め休止となるPEのローカルメモリをアクセスする稼動PEの情報を与えておき、同情報と識別手段の情報から、稼動するPEは休止しているPEのローカルメモリに直接アクセスし、同ローカルメモリを稼動するPEのローカルメモリの一部として使用する。

【0072】さらに、バイパス手段とアクセス手段を、同一チップ内に納められた複数PEに適用する事により、高速な動作を保証すると共に、バイパス手段及びアクセス手段をPEの上に配線することにより配線面積を小さくし、チップ面積の縮小を行う。

【0073】第4の発明の並列計算機によれば、メッセージの出力先の決定後にフリットをバッファに蓄えるので、出力先チャンネルが空いていれば、通信が停止している他のメッセージによってそのメッセージの通信が妨害されることはない。したがって、並列計算機全体として通信時間は短縮され、スループットは向上する。

【0074】第5の発明の並列計算機によれば、あるプロセッサが隣接するプロセッサからメッセージを受け取り、そのメッセージが自プロセッサ宛でなければ、通信チャンネルと制御信号をバイパスする。メッセージの中継を行うプロセッサで、メッセージおよび制御信号をバイパスし、通信を行っているプロセッサ間の距離を実際よりも小さくしている。

【0075】

【実施例】以下、図面を参照しながら本発明の実施例を説明する。

【0076】第1の発明

第1の発明の一実施例の構成を図1に示す。図1に示した並列計算機は、 n 個（ n は整数）のプロセッサ1からなるプロセッサアレイと、プロセッサアレイに対して命令を供給するプロセッサアレイ制御装置2とからなる。図示していないが、プロセッサ1間およびプロセッサ1とプロセッサアレイ制御装置2間には、データ転送と制御のための通信ネットワークが存在する。

【0077】なお、今回の実施例では、プロセッサアレイ制御装置2が2種類の命令列を供給する場合について示すが、第1の発明はこれに限らず、複数種類の命令列を供給することができるものである。

【0078】プロセッサアレイ制御装置2は、命令メモリを持ち、そこから命令を取り出し、プロセッサアレイに対して1度に2個の命令を供給することができる。各プロセッサ1は、プロセッサアレイ制御装置2から同時に供給された2個の命令のうちの1つの命令を受け取り実行する。

【0079】具体的には、プロセッサ番号が偶数のプロセッサ1はプロセッサアレイ制御装置2の出力端子OUT0から出力される命令列の命令を受け取り実行し、プロセッサ番号が奇数のプロセッサ1は出力端子OUT1から出力される命令列の命令を受け取り実行する。

【0080】各プロセッサ1は、演算部、レジスタファイル、制御部、ローカルメモリ、メモリインターフェース部、プロセッサ間通信部からなる。これらのプロセッサ1には2つの動作モードがある。1つは、各プロセッサ1が独立に動作するレジスタ非共有モードであり、もう1つは、隣のプロセッサ1とレジスタを共有するレジスタ共有モードである。

【0081】ここでいうレジスタの共有とは、自分のプロセッサ1内のレジスタの内容が対応する相手のレジスタの内容と常に同じようになっているということである。つまり相手のプロセッサ1内のレジスタにデータを書き込んだ後に、自分の対応するレジスタを読み出すと相手のプロセッサ1が書き込んだデータと同じデータが読み出せるというものである。

【0082】プロセッサ番号が偶数のプロセッサ1は、自分のプロセッサ番号より1つ大きい番号をもつプロセッサ1とレジスタを共有し、プロセッサ番号が奇数のプロセッサ1は、自分のプロセッサ番号より1つ小さいプロセッサ1とレジスタを共有する。

【0083】まず、プロセッサ数よりも並列度が大きい処理を実行する場合について説明する。この場合には、プロセッサ1の動作モードはレジスタ非共有モードとし、すべてのプロセッサ1に同じ処理を実行させるために同じ命令を供給するようにする。そのためには、この実施例では、プロセッサ1ごとに供給される命令列は固定されているので、プロセッサアレイ制御装置2の2つの出力端子OUT0、1から同一の命令列を供給することにより、1つの命令列を2つの命令列として供給するように制御する。

【0084】次にプロセッサ数よりも並列度が小さい処理を実行する場合について説明する。この場合には、プロセッサ1の動作モードはレジスタ共有モードとする。プロセッサ番号が偶数のプロセッサ1とその番号より1つ大きい番号のプロセッサ1とが1つのクラスタを構成する。

【0085】クラスタ内の2つのプロセッサ1には、プロセッサアレイ制御装置2によって抽出された並列実行可能な2つの命令が出力端子OUT0、1からそれぞれのプロセッサ1に同時に供給され、それぞれのプロセッサ1では供給された各命令を並列に実行する。

【0086】ここで、図2のような式の計算をする場合について説明する。例えば、プロセッサ数 n が1000であるとする。

【0087】この計算を1000組のデータに対して行なう場合には、プロセッサ1のモードをレジスタ非共有

モードにし、プロセッサアレイ制御装置2の出力端子OUT0、1から、図3(a)、(b)のような同一な命令列1、2を供給する。各プロセッサ1に1組ずつのデータの処理を割当てることにより、1個のプロセッサ1で1000組のデータを処理するのに比べ1000分の1の時間で処理できる。

【0088】しかし、このレジスタ共有モードで500組のデータを処理する場合は、1個のプロセッサ1で500組のデータを処理するのと比べ、プロセッサ数が1000であるのに500分の1の時間でしかなく、並列計算機的能力を十分に生かしきれない。

【0089】一方、プロセッサ1のモードをレジスタ共有モードにし、プロセッサアレイ制御装置2から図4(a)、(b)のような並列実行可能な2種類の命令列1、2を出力端子OUT0、1からそれぞれ供給すると、2台のプロセッサ1で1組のデータを処理することになり、1000分の1の時間で実行することが可能である。

【0090】図3の命令列では、1回の処理を行なうのに4命令分実行時間がかかるが、図4の場合には、2命令分の実行時間ですみ、実行時間が半分になる。したがって、500組のデータに対する処理の場合でも図4の命令列で実行させれば、1個のプロセッサ1で処理する場合に比べて1000分の1の時間で実行することが可能である。

【0091】なお、プロセッサアレイ制御装置2が供給する並列実行可能な複数の命令は、もともとメモリ上に並列実行可能な複数の命令列を用意しておき、それを供給するという方法と、単一の命令列を用意しプロセッサアレイ制御装置2内で並列実行可能な命令を抽出して複数命令を供給するという方法が考えられるがどちらでも良い。後者の場合には、通常のSIMD型並列計算機とのオブジェクト互換性が保たれることになるため、その点は有利である。

【0092】図1に示した実施例では、複数供給される命令列のうちのどちらを各プロセッサ1が受け取るかが固定されているが、図5に示すように、いずれの命令列を受け取るかをセクタ3で選択できるようにする構成もある。この構成では、すべてのプロセッサ1は、プロセッサアレイ制御装置2から供給される2つの命令列のどちらでも選択して受け取ることができる。

【0093】その選択は各プロセッサ1の命令入力の前におかれているセクタ3で行なわれ、どちらを選ぶかは、あらかじめ設定することもできるし、各プロセッサ1の演算結果からどちらを選択するかを決定することもできる。クラスタを構成する際には、並列実行可能な2つの命令列を受け取った2つのプロセッサ1で構成すれば良い。

【0094】この場合、if-then-elseのような処理を実行するときに、then処理とelse処

理を並列に実行するように2つの命令列を出力端子OUT0、1からそれぞれ供給し、各プロセッサ1はif文の条件によりいずれかの命令列を選択するようにすれば、if-then-elseの並列実行が可能である。

【0095】superscalarあるいはVLIWなどの複数命令の並列実行機構を持ったプロセッサなどを要素プロセッサとして用いる場合には、並列度の高い問題を解く場合でも、問題の並列度を犠牲にして複数命令の並列実行をすることになり、必ずしも効果があがるとはかぎらないが、この第1の発明では、並列度の高い問題では、問題の並列性を利用して解くことができるので、高速に解くことが可能である。

【0096】第2の発明

図6は、第2の発明の並列計算機に係わる一実施例の構成を示すブロック図である。

【0097】同図に示す並列計算機は、大きく分けて、1台の制御プロセッサ(CP)11と、複数台の演算プロセッサ(PE)12から構成される。制御プロセッサ11には全体メモリ(M)13が接続され、演算プロセッサ12にはそれぞれローカルメモリ(LM)14が接続される。制御プロセッサ11と各演算プロセッサ12は、グローバルバス15、および同期バス17で接続される。演算プロセッサ12は、ネットワーク16で相互接続される。ネットワーク16の形態としては、格子結合、2進nキューブ結合、木結合等、多くのものが知られており、いずれの方式でもよい。

【0098】制御プロセッサ11は、それ自体が一つの情報処理装置であり、全体メモリ13に格納されているプログラム(命令列)を読み込み、解読し、解読結果にしたがって、全体メモリ13に格納されているデータに対して、処理を行う機能を有する。さらに制御プロセッサ11には、演算プロセッサ12に対して制御を行う機能が追加されている。

【0099】全体メモリ13に格納されている制御プロセッサ11のプログラムには、制御プロセッサ11自身での処理を指示する命令の他に、演算プロセッサ12での処理を指示する命令が含まれている。

【0100】制御プロセッサ11は、メモリ13から読み出した命令を解読し、制御プロセッサ11で実行すべき命令であれば、制御プロセッサ11で実行する。解読した命令が演算プロセッサ12で実行すべき命令であれば、グローバルバス15を介して全演算プロセッサ12に放送する。ここで放送する命令は、解読済みのもの(マイクロ命令)でも解読前のもの(機械語命令)でもよいが、ここでは解読済みの命令を放送するものとして話を進める。

【0101】制御プロセッサ11は、グローバルバス15を介して、すべての、あるいは特定の演算プロセッサ12から、データの読み出し、および書き込みを行うこ

とができる。また、同期バス17を介して、すべての演算プロセッサ12の状態の論理和を認識することができる。

【0102】さらに制御プロセッサ11は、フロントエンド計算機18と結合されている。フロントエンド計算機18では、制御プロセッサ11および演算プロセッサ12で実行するプログラムの開発を行ったり、処理すべきデータの入出力を行う。

【0103】フロントエンド計算機18と制御プロセッサ11は、実行時、制御プロセッサ11で実行すべきプログラムをフロントエンド計算機18から全体メモリ13に転送する。さらに、制御プロセッサ11で実行すべきデータをフロントエンド計算機18から全体メモリ13に転送したり、演算プロセッサ12で実行すべきデータを、フロントエンド計算機18から各演算プロセッサ12に分配したり、あるいは処理結果を各演算プロセッサ12や全体メモリ13からフロントエンド計算機18に転送したりする機能を有する。

【0104】演算プロセッサ12は、制御プロセッサ11からグローバルバス15を介して放送される命令にしたがって、自ローカルメモリ14のデータに対して演算を行ったり、ネットワーク16を介して他の演算プロセッサ12とデータ通信を行ったりして処理を行う。

【0105】第2の発明の中心となる演算プロセッサ12は、図7に示すような構成になっている。

【0106】まず、ローカルメモリ14と演算プロセッサ12とは、メモリアドレスバス21と、メモリデータバス22によって接続されている。

【0107】マルチプレクサ(MUX)23は、第2の発明において追加されたものであり、命令レジスタ24に格納する命令として、制御プロセッサ11からグローバルバス15を介して放送される命令か、ローカルメモリ14から読み出されたデータかを選択する。マルチプレクサ23の一方の入力は、制御プロセッサ11からのグローバルバス15が、もう一方の入力には、メモリデータバス22がそれぞれ接続されている。

【0108】命令レジスタ24には、マルチプレクサ23で選択された命令が格納される。演算プロセッサ12を構成する他のすべての要素は、命令レジスタ24からの指示によって制御される。ここでは説明を容易にするために、命令レジスタ24の最上位ビットでマルチプレクサ23の制御を行うこととし、最上位ビットが0ならば、制御プロセッサ11から放送される命令が命令レジスタ24に格納され、最上位ビットが1ならば、ローカルメモリ14から読み出されるデータが命令レジスタ24に格納されるものと仮定して話を進める。

【0109】プログラムカウンタ(PC)25は、第2の発明をより効果的にするために追加されたものであり、ローカルメモリ14に格納されているMIMD処理用の命令列を読み出すために用いられる。プログラムカ

ウンタにMIMD用命令列のアドレスを格納し、読み出し操作を行うことによって、メモリデータバス22に命令が読み出される。命令列を連続して読み出すために、プログラムカウンタ25には、自分自身の値をインクリメントする機能を持つ。

【0110】アドレスレジスタ(AR)26、データレジスタ(DR)27は、ローカルメモリ14をアクセスするために用いられる。アドレスレジスタ26は、内部バス28から書き込まれ、メモリアドレスバス21に値を出力する。データレジスタ27は、メモリデータバス22と内部バス28との間に置かれる双方向のレジスタである。

【0111】データ読み出し時は、内部バス28からアドレスレジスタ26に、読み出すべきメモリ13のアドレスが格納され、ローカルメモリ14に対して読み出し操作を行い、読み出されたデータはメモリデータバス22からデータレジスタ27に格納される。データ書き込み時は、アドレスレジスタ26に書き込むべきメモリ13のアドレスを格納し、データレジスタ27に書き込みデータを格納し、書き込み操作を行うことにより、書き込みが行われる。

【0112】ALU29は、内部バス28を介して、レジスタファイル30に格納されるデータに対して、算術あるいは論理演算を行う。

【0113】状態制御レジスタ(PSW: Processor Status Word)31は、ALU29での演算状態を表示するビットと、演算プロセッサ12での演算を制御するビットから構成される。演算を制御するビットの内訳は次の通りである。このうち、(1)~(3)は、演算のALU29で演算が行われるたびに設定される。(4)、(5)は、直接、状態制御レジスタ31を書き換えることによって設定する。

【0114】(1)ゼロビット: 演算結果がゼロになるとセットされる。

【0115】(2)正ビット: 演算結果が正のときにセットされる。

【0116】(3)オーバーフロービット: 演算でオーバーフローが発生したときにセットされる。

【0117】(4)同期ビット: 全演算プロセッサ12の同期ビットはワイアード・オア接続され、制御プロセッサ11に接続される。同期を取る時点で演算プロセッサ12がこのビットをセットすると、制御プロセッサ11は、全演算プロセッサ12が同期ビットをセットしたか否かを認識できる。

【0118】(5)マスクビット: このビットがセットされると、以後、制御プロセッサ11からマスク解除命令が発行されるまで、命令レジスタ24の内容にかかわらず、演算プロセッサ12は動作しない。

【0119】プロセッサ番号レジスタ(PNR: Processor Number Register)32は、各演算プロセッサ固有

のプロセッサ番号が格納されるレジスタである。このレジスタを読み出すことによって、自分のプロセッサ番号を知ることができる。このレジスタはハードウェア的に設定する方法や、システム立ち上がり時に制御プロセッサ 11 が設定する方法等が考えられる。ここではハードウェア的に設定されているものとするが、他の方法でも差し支えない。

【0120】通信制御部 33 は、ネットワーク 16 を介して他の演算プロセッサ 12 と通信したり、グローバルバス 15 を介して制御プロセッサ 11 との通信を行うためのレジスタや外部の通信プロトコル制御装置から構成される。演算プロセッサ 12 は、この通信制御部 33 によって、制御プロセッサ 11 や他の演算プロセッサ 12 と通信を行うことができる。

【0121】命令レジスタ 24 への命令の格納は、前述したように、次の 2 通りの方法で行われる。第一の方法は、制御プロセッサ 11 より放送されるものである。制御プロセッサ 11 から全演算プロセッサ 12 に同一の命令が放送されるため、マスクビットがセットされていない演算プロセッサ 12 は、すべて同じ処理を行う。すなわち、SIMD 方式の並列処理が行われる。

【0122】第 2 の方法は、ローカルメモリ 14 にあらかじめ格納されている命令を読み出して、命令レジスタ 24 に格納するものである。この命令は、あらかじめ各演算プロセッサ 12 で異なったものを格納しておくことにより、各演算プロセッサ 12 で異なった処理を行うことができる。すなわち、MIMD 方式の並列処理となる。

【0123】以下では、SIMD 方式と MIMD 方式の 2 つの処理方式、およびこの 2 つの処理方式間の以降を中心に、本並列計算機での処理手順について説明する。

【0124】まず、第 2 の発明の並列計算機で実行するプログラムについて説明する。このプログラムは、大きく次の 3 つに分けられる

(1) 逐次処理される部分。

【0125】ループ制御等、並列処理できない部分で、制御プロセッサ 11 で実行される。

(2) 各演算プロセッサ 12 が一斉に処理を行う部分 (SIMD 処理)。

【0126】制御プロセッサ 11 により、各演算プロセッサ 12 が同じ処理を行う。

【0127】(3) 各演算プロセッサ 12 で異なった処理を行う部分 (MIMD 処理)。

【0128】制御プロセッサ 11 の制御を離れて、各演算プロセッサ 12 が独自に実行する。

【0129】このうち、(1) と (2) は、制御プロセッサ用プログラムとしてコンパイルされ、機械語命令列が全体メモリ 13 にロードされる。そして、制御プロセッサ 11 が解読、実行あるいは実行制御を行う。(3) は各演算プロセッサ用としてコンパイルされ、さらにマ

イクロ命令列に展開された後、実行される演算プロセッサ 12 のローカルメモリ 14 にロードされる。

【0130】次に、初期データのマッピング処理について説明する。

【0131】全体メモリ 13 に、制御プロセッサ 11 で実行される命令とデータがフロントエンド計算機 18 から転送される。

【0132】次に、各演算プロセッサ 12 のローカルメモリ 14 に、演算データと、MIMD 動作されるべき命令とが格納される。これらのデータは、各演算プロセッサ 12 によって異なるため、演算プロセッサ 12 ごとに逐次的に行われる。処理手順は次の通りである。

【0133】(1) 制御プロセッサ 11 は、全演算プロセッサ 12 に対して、データを転送する演算プロセッサ番号と、マスク命令を放送する。

【0134】(2) 各演算プロセッサ 12 は、制御プロセッサ 11 からの命令に従って、制御プロセッサ 11 から放送されるプロセッサ番号と、自分のプロセッサ番号とを比較し、一致していなければマスクビットをセットする。したがって以後の処理は、制御プロセッサ 11 が指定した番号の演算プロセッサ 12 のみで行われる。

【0135】(3) 制御プロセッサ 11 は、データを格納するべきローカルメモリ 14 の領域の先頭アドレスを演算プロセッサ 12 に転送し、DMA モードと、制御プロセッサ 11 から転送するデータのメモリ 13 への書き込みを指示する命令を発行する。

【0136】(4) 制御プロセッサ 11 は、指定した演算プロセッサ 12 に転送するデータをフロントエンド計算機 18 から取得し、グローバルバス 15 を用いて演算プロセッサ 12 に順次転送する。

【0137】(5) 指定された演算プロセッサ 12 は、制御プロセッサ 11 から転送されるデータを、指定されたローカルメモリ 14 の領域に、DMA モードで順次書き込む。

【0138】(6) 転送が終了すると、制御プロセッサ 11 は、DMA 中止命令と、マスク解除命令を全演算プロセッサ 12 に対して発行する。

【0139】(7) 以上で述べた (1) ~ (6) の処理を全演算プロセッサ 12 に対して行う。

【0140】MIMD 方式の並列処理を行うにあたって、MIMD 用命令アドレステーブルを作成する必要がある。このアドレステーブルの作成方法を下記に示す。

【0141】MIMD 動作される命令列は、初期データマッピング時は、演算データと区別されることなくローカルメモリ 14 に格納される。連続した命令列は、ローカルメモリ 14 の連続したアドレスに格納されることが望ましい。また、同一時刻に MIMD 処理される命令列の開始アドレスは、各演算プロセッサ 12 で同じであることが望ましいが、異なっても実行可能である。

【0142】MIMD 動作される命令列の開始アドレス

は、各演算プロセッサ12で同一の場合は制御プロセッサ11または演算プロセッサ12が、各演算プロセッサ12で異なる場合は演算プロセッサ12が認識しておく必要がある。

【0143】図8に、各演算プロセッサ12でMIMD動作命令開始アドレスが同一の場合の制御プロセッサ11のアドレステーブルの一例を示す。アドレステーブルはどの領域に作成してもよいが、ここでは全体メモリ13の0番地から作成されるものと仮定する。同一時期にMIMD動作される命令列の開始アドレスが、各演算プロセッサ12で異なる場合は、各演算プロセッサ12がローカルメモリ14中にアドレステーブルを作成する必要がある。

【0144】以下に、SIMD方式による処理を説明する。

【0145】第2の発明の並列計算機における制御プロセッサ11から放送される命令により、全演算プロセッサ12が同一の処理を行う方式である。制御プロセッサ11は、全体メモリ13から命令を読み込み、解読した結果、それが演算プロセッサ12で実行すべきものならば、グローバルバス15を介して全演算プロセッサ12に放送する。各演算プロセッサ12では、制御プロセッサ11から放送される命令を命令レジスタ24に入力するように、マルチプレクサ23を制御する。

【0146】マルチプレクサ23の制御は、命令レジスタ24の最上位ビットで行われ、最上位ビットが0ならば、制御プロセッサ11から放送される命令が命令レジスタ24に格納される。したがって、制御プロセッサ11から放送されるSIMD処理用の命令は、最上位ビットは常に‘0’となるように制御プロセッサ11を設計する必要がある。命令レジスタ24の各ビットが、演算プロセッサ12の他の構成要素を制御することによって処理が行われる。

【0147】次に、SIMD方式からMIMD方式へ移行する処理手順を説明する。

【0148】制御プロセッサ11の命令には、各演算プロセッサ12のローカルメモリ14に格納されている命令列に実行を移行するための命令が用意されている。この命令は、図8に示すアドレステーブルの番号を指定するフィールドを持つ。この命令が制御プロセッサ11で実行されると、次のような手順で各演算プロセッサ12のMIMD用命令列に処理が移行される。

【0149】(1) アドレステーブルから命令列の開始アドレス読み出され、グローバルバス15を介して全演算プロセッサ12に放送される。

【0150】(2) 制御プロセッサ11からの命令による制御で、各演算プロセッサ12で、(1)で転送されたアドレスが、プログラムカウンタ25に格納される。

【0151】(3) 制御プロセッサ11からの命令による制御で、各演算プロセッサ12で、プログラムカウンタ

25で示されるローカルメモリ14のアドレスから、MIMD処理のための命令が読み出される。このときの制御プロセッサ11からの命令は、最上位ビットが‘1’になっている。したがって、次の命令レジスタ24の入力は、ローカルメモリ14から読み出されたMIMD用命令が選択されるよう、マルチプレクサ23が切り換わる。

【0152】(4) ローカルメモリ14から読み出された命令は、命令レジスタ24に格納され、命令レジスタ24からの指示によって処理が行われる。この処理は、各演算プロセッサ12で異なっても良いため、MIMD処理となる。

【0153】次に、MIMD方式に移行した後の処理を以下に示す。

【0154】前述したように、MIMD処理が開始されたとき、プログラムカウンタ25には、MIMD用命令列の先頭アドレスが格納されている。プログラムカウンタ25は、命令レジスタ24からの制御によってインクリメントされる機能を備えている。これによって、ローカルメモリ14に格納されたMIMD用命令列を順次読み出すことができる。

【0155】また、命令レジスタ24に格納される命令の選択は、現在の命令レジスタ24の最上位ビットで決定される。すなわち最上位ビットが‘1’ならば、ローカルメモリ14から読み出した命令が命令レジスタ24に格納される。

【0156】したがって、ローカルメモリ14に格納されるMIMD処理用命令列は、最上位ビットを‘1’にし、さらにプログラムカウンタ25をインクリメントしながらローカルメモリ14を読み出すように制御コードを設定しておけば、ローカルメモリ14の命令列を順次命令レジスタ24にロードして、MIMD処理が可能である。命令列の最後尾は、最上位ビットを‘0’にしておけば、MIMD処理終了後、制御プロセッサ11からの命令が命令レジスタ24にロードされるように、マルチプレクサ23が制御される。

【0157】もちろん、単一の命令に、上記の最後尾の命令の設定を行っておけば、単一の命令のMIMD動作も可能である。

【0158】最後に、MIMD方式からSIMD方式への移行処理を説明する。

【0159】MIMD処理の終了は、各演算プロセッサ12でまちまちであるため、制御プロセッサ11は、全演算プロセッサ12でのMIMD処理の終了を確認してからSIMD処理を開始する必要がある。

【0160】(1) 制御プロセッサ11は、各演算プロセッサ12でのMIMD処理を起動した後、待ち状態に入る。この状態では、制御プロセッサ11は、演算プロセッサ12に対してNOP (No Operation) コードを発行し続ける。

【0161】(2) 演算プロセッサ12はMIMD処理を行うが、MIMD処理用命令列の最後尾の命令として、状態制御レジスタ31の同期ビットをセットし、さらに最上位ビットを‘0’にした命令を挿入しておく。

【0162】(3) 各演算プロセッサ12では、MIMD処理用命令列の最後のコードを実行すると、状態制御レジスタ31の同期ビットがセットされ、さらに命令レジスタ24への入力が、以後は制御プロセッサ11が発行する命令を選択するようにマルチプレクサ23が切り換わる。各演算プロセッサ12の同期ビットはワイヤード・オア接続され、制御プロセッサ11は、全演算プロセッサ12で同期ビットが1にセットされたか否かを認識できる。

【0163】(4) 制御プロセッサ11は、(1)で演算プロセッサ12のMIMD処理を起動した後、全演算プロセッサ12で同期ビットに1がセットされたことを認識するまでNOP命令を発行し続ける。早く処理が終わった演算プロセッサ12は、制御プロセッサ11からNOPコードを受け取って実行するが、副作用はない。

(5) 制御プロセッサ11は、全演算プロセッサ12の同期ビットに1がセットされたこと、すなわち全演算プロセッサ12でMIMD処理が終了したことを確認したら、NOPコードの発行を中止し、SIMD処理用の命令の発行を開始する。

【0164】(6) 以降は、制御プロセッサ11からの命令にしたがって、全演算プロセッサ12でSIMD処理が行われる。

【0165】以上で述べたように、SIMD型並列計算機にわずかなハードウェア(マルチプレクサ23、プログラムカウンタ25)を追加するだけで、MIMD動作が可能となる。

【0166】なお、以上の実施例は、第2の発明を実現するための一例を示したものであり、これに限定されるものではない。例えば、命令レジスタ24への入力の選択は、命令レジスタ24の最上位の1ビットを用いて行ったが、状態制御レジスタ31の1ビットを用いて行うようにすることもできる。この場合の構成図を図9に示す。

【0167】図7と異なる点は、命令レジスタ24への入力データを選択するマルチプレクサ23の制御を、状態制御レジスタ31の1ビットを用いて行うことである。このような構成で、MIMD動作を行うための手順を次に示す。なお、図7の実施例と共通の処理は、記述を省略した。

【0168】1. SIMD処理

状態制御レジスタ31の、マルチプレクサ23を制御するビットを、制御プロセッサ11からの命令を命令レジスタ24に入力するように設定する。以後、状態制御レジスタ31のマルチプレクサ23を制御するビットを書き換ええない限り、演算プロセッサ12は制御プロセッサ

11からの命令にしたがって処理を行う。

【0169】2. MIMD処理への移行

制御プロセッサ11の命令には、各演算プロセッサ12のローカルメモリ14に格納されている命令列に実行を移行するための命令が用意されている。この命令は、図8に示すような命令列の開始アドレスの格納されている番地を指定するフィールドを持つ。この命令が制御プロセッサ11で実行されると、次のような手順で各演算プロセッサ12の命令列に処理が移行される。

【0170】(1) 指定されたアドレステーブルに格納されている命令列の開始アドレスがグローバルバス15を介して全演算プロセッサ12に放送される。

【0171】(2) 制御プロセッサ11からの命令による制御で、各演算プロセッサ12で、(1)で放送されたMIMD用命令列の開始アドレスが、プログラムカウンタ25に格納される。

【0172】(3) 制御プロセッサ11からの命令による制御で、各演算プロセッサ12で、プログラムカウンタ25で示されるローカルメモリ14の開始アドレスから、MIMD処理のための命令が読み出され、命令レジスタ24に格納される。

【0173】(4) 状態制御レジスタ31の、マルチプレクサ23を制御するビットを、命令レジスタ24への入力データとして、ローカルメモリ14から読み出したデータを選択するように設定する。

【0174】(5) 命令レジスタ24からの指示によって処理が行われる。この処理は、各演算プロセッサ12で異なっても良いため、MIMD処理となる。

【0175】3. MIMD方式による処理

前実施例と同様に、プログラムカウンタ25には、MIMD用命令列の先頭アドレスが格納されており、この値をインクリメントさせながら、ローカルメモリ14から命令列を順次読み出すことができる。状態制御レジスタの設定により、ローカルメモリ14から読み出された命令列は、命令レジスタ24に格納されるようにマルチプレクサ23が制御される。これによって、ローカルメモリ14に格納された命令列を順次読み出し、実行することができる。

【0176】4. MIMD方式からSIMD方式への移行

MIMD処理の終了は、各演算プロセッサ12でまちまちであるため、制御プロセッサ11は、全演算プロセッサ12でのMIMD処理の終了を確認してからSIMD処理を開始する必要がある。

【0177】(1) 制御プロセッサ11は、MIMD処理を起動した後、待ち状態に入る。この状態では、制御プロセッサ11は、演算プロセッサ12に対してNOP(No Operation)コードを発行する。

【0178】(2) 各演算プロセッサ12のMIMD処理用命令列の最後尾に、状態制御レジスタ31の同期ビ

ットをセットし、かつ、マルチプレクサ23を制御するビットを制御プロセッサ11からの命令を命令レジスタ24に入力するように設定する命令を挿入しておく。

【0179】(3)各演算プロセッサ12では、MIMD処理用命令列の最後のコードを実行すると、状態制御レジスタ31の同期ビットがセットされ、さらに命令レジスタ24への入力が、以後は制御プロセッサ11が放送する命令を入力するようにマルチプレクサ23が切り換わる。

【0180】(4)制御プロセッサ11は、(1)で演算プロセッサ12のMIMD処理を起動した後、全演算プロセッサ12で同期ビットに1がセットされたことを認識するまでNOP命令を発行し続ける。早く処理が終わった演算プロセッサ12は、制御プロセッサ11からNOPコードを受け取って実行するが、副作用はない。

(5)制御プロセッサ11は、全演算プロセッサ12の同期ビットに1がセットされたこと、すなわち全演算プロセッサ12でMIMD処理が終了したことを確認したら、NOPコードの発行を中止し、SIMD処理用の命令の発行を開始する。

【0181】(6)以降は、制御プロセッサ11からの命令にしたがって、全演算プロセッサ12でSIMD処理が行われる。

【0182】このように、命令レジスタ24への命令の入力の選択を、状態制御レジスタによって設定することもできる。この方法では、制御レジスタの設定のために命令が必要となるが、演算結果によって、制御プロセッサ11からの命令列か、メモリ13に格納されている命令列かのどちらかを実行するかを選択することも可能で、より柔軟性の高い処理が実現できる。

【0183】以上で述べた実施例では、制御プロセッサ11は解読済みの命令、すなわちマイクロ命令を各演算プロセッサ12に放送する方法を示した。これに対して、解読する前の機械語命令をそのまま放送することも可能である。この場合、各演算プロセッサ12に、機械語を解読したマイクロ命令を発生するための機能を設ける必要があり、ハードウェア量は増加する。

【0184】しかし、近年開発されているRISC (Reduced Instruction Set Computer) プロセッサでは、機械語の解読は比較的簡単に行われるため、それほどハードウェア量は増加しないと考えられる。もちろん、このような場合でも、第2の発明で示したように、SIMD処理とMIMD処理の両方を可能にする構成が可能である。

【0185】第3の発明

図10は、第3の発明の並列計算機に係わる一実施例の構成を示す図である。同図には、簡単なモデルとして1次元リング結合の並列計算機を示した。この図に示す通り、16個のPE0~15がリング状に配置され、各々のPE0~15は隣のPEと通信を行うものとする。

【0186】問題サイズが16の倍数の時、分割されたタスクのPEへの割り付けは容易である。すなわち例えばサイズが16の場合、PE0には1番目のタスクを割り当て、PE1には2番目のタスクの割り付けを行い、PE15には16番目のタスクを割り付ける。この場合、各PEに均等にタスクが割り付けられ、バランス良く処理を実行できる。

【0187】しかし、問題サイズが10の場合、16台のPE0~15の内の10台にタスクを割り付けるが、例えばPE0からPE9までに割り付けるとする。この時PE10からPE15までは休止となるので、識別回路にPE0からPE9までが稼働であり、PE10からPE15までが休止である情報をセットする。各PEのバイパス回路は識別回路からの情報により通信時のバイパスを行う。

【0188】本例ではPE10からPE15のバイパス回路がバイパスを行う。これによりソフトから見たハードウェアの構成はPE0からPE9までがリング状に接続された10台並列の1次元接続の並列計算機となる。左から順にPE0、PE1...PE9と並んでいる場合、PE9の右の通信バスはPE0の左側の通信バスに接続される。

【0189】図11、12にバイパス回路の詳解図を示す。図11はPE間が双方向バスで通信を行う場合のアドレス/データバスである。バイパス経路はPEを飛び越すように、PEの右側バスと左側バスに接続する。双方向バスには双方向3-stateバッファ41を挿入する。3-ステートバッファには識別回路42から自PEの稼働/休止情報が送られる。PEが稼働の場合は3-stateとなり、休止の場合はバッファがオープンとなり、通信可能となる。

【0190】双方向バッファ41の方向制御は並列計算機の仕様による。例えばSIMD型並列計算機で、同期的に処理を実行する場合、データの転送方向は全てのPEで同じであるため、PEの制御を行うACU (Array Control Unit) から転送方向を指定するだけでよい。

【0191】一方、MIMD型並列計算機では、隣接通信は非同期である場合が多く、そのためハンドシェイクを行う必要がある。ハンドシェイクには要求信号と許可信号の信号線が必要となる。この場合、双方向バッファ41の方向は、アクセス要求を出した側とアクセスが読み出ししか書き込みかに依存してくる。そのため、バッファ41の方向はバッファ41に付属の制御回路で要求信号とRead/Write信号から決定する。

【0192】MIMD型並列計算機で必要となる要求信号、許可信号、Read/Write信号などは一方の信号である場合が多い。特に今問題としているPE間での転送は通信相手が固定できるためなおさらである。通信方向が1方向の場合の詳解図が図12である。例え

ば右方向への要求信号を考えると、自PEの左のPEから自PEに向けられた要求信号と、自PEから右のPEに送る要求信号を識別回路42からの稼動/休止の信号により、セクタ43aで選択して右のPEへ送る。

【0193】例えば自PEが稼動の場合、右のPEに送る要求信号は自PEからの要求信号をセクタ43aで選択して送り、自PEが休止している場合は左のPEから自PEに送られてくる要求信号を右のPEに送る。これにより、自PEが休止の場合は左のPEからの要求信号は直接右のPEに送られることになり、バイパスを行うことが出来る。

【0194】上記例ではPE10からPE15のかたまりを休止とした。しかし、このように局地的にかたまって休止にした場合、バイパスする距離が大きくなる。上記例では6PE分の距離をバイパスすることになる。図11の場合、バイパスの際にはバッファ41を通過する。連続してバイパスする数が増えればそれだけ通過するバッファ41の数は増え、なおかつ物理的な距離も遠くなり、通信の為の時間が増加してしまう。

【0195】上記MIMD型並列計算機でのハンドシェイク方式ならば、多少通信時間に誤差が生じても防ぐことが出来るが、SIMD型並列計算機での同期方式では、各PE間での通信所要時間にばらつきがあると、問題が生ずる。

【0196】実際には、転送命令が40nsで実行しなければならず、隣接PE間で転送する場合25ns必要な場合、バイパス1段を通るのに5nsの遅延が生ずるとすると、 $15 \div 5$ で最大3PE分のバイパスしか許されず、上記例での6個のPEのバイパスは不可能となる。

【0197】そのためバイパスするPEを選択する必要がある。例えば休止にするPEを1つおきにすれば、PE間には1PE分の遅延しか生じなくなる。上記例では休止するPEを例えばPE5、PE7、PE9、PE11、PE13、PE15と選択することにより可能となる。上記遅延の制約がある場合、上記16台並列の計算機では、4台並列から、16台並列まで自由に並列台数を変えることの出来る1次元リング結合並列計算機が実現できる。

【0198】また、上記例では各PE0~15の保有するPE番号(0~15)を固定としているが、これだと休止するPEがある場合、稼動PEの番号が連続とならない。実行するプログラムが連続したPE番号を必要とする場合は、PE番号を、稼動するPEに与えるデータの一変数の値とすることで連続したPE番号とすることができる。

【0199】第3の発明は仮想プロセッサの概念にも応用できる。すなわち、16台並列のPE0~15に問題サイズが16以上のタスクを割り付ける場合を考える。タスクの数が16の倍数の場合、各PE0~15に均等

に割り付けることが出来る。例えば32のタスクを割り付ける場合、各PE0~15に2タスクずつ割り付けられ、通常の仮想プロセッサの処理で何問題ない。

【0200】すなわち各PE内部の2つのタスクを処理するとき、それぞれのタスクを処理するために2つの状態を示す状態保持手段を用い、タスク1とタスク2を各々実行するとき上記状態を切り替えることにより、現在どちらの処理を実行しているかを認識し、仮想プロセッサを実現し、16台で32のタスクを処理する。

【0201】しかし、例えばタスク数が26の場合、各PE0~15に均等に割り付けようとすると、タスクを1個受け持つPEと、2個受け持つPEとができる。各PEが受け持つ2つのタスクをタスク1とタスク2とすると、タスク1を実行する場合は16台のPEが一斉に行えば良いが、タスク2を実行する際は10台のPEが稼動となり、残り6台のPEが休止となる。

【0202】つまりタスク2を実行する際は、上記例での10個のタスクを実行する場合と同様の状態となる。そのため、タスク1を実行する場合は16台全てが稼動し、タスク2を実行する場合はバイパス回路を使用して10台並列の計算機に切り替えて実行を行えば仮想マシンとして26台並列の並列計算機として使用できる。バイパス回路の使用/非使用は、上記状態保持手段に、上記識別回路42を連動することにより実現できる。すなわちタスク1を処理する状態の時、識別回路42は全てのPE0~15が稼動であるとしてバイパス回路を用いず、タスク2を処理する状態の時、識別回路42は与えられているPEの使用/非使用の情報からPEの稼動/休止を決定し、バイパス回路の使用/非使用を決定する。

【0203】次に2次元格子トラス結合の場合を示す。基本的には1次元リング結合と同様であり、バイパスの方向が列方向と行方向の2方向になる。このため、図11、12で示した双方向バッファ41あるいはセクタ43も、列方向と行方向の2方向分必要となる。制御の簡単化を考えると、1次元リング結合で1PEが休止の対象となったのに対し、2次元格子トラス結合では行あるいは列が休止の対象となる。

【0204】図13に8×8のアレイに問題サイズが6×5の処理を割り付けた場合を示す。まず行方向に関しては、8行のPE列のうち、6行を稼動とする。各PEを(X, Y)で表すとすると、1次元リング結合の場合と同様に、局所的に休止PEを固めないようにすると、PE(0, Y)、PE(1, Y)、PE(2, Y)、PE(3, Y)、PE(4, Y)、PE(6, Y)の6行がそれぞれ稼動となる(図中、丸印)。

【0205】列方向に関してはPE(X, 0)、PE(X, 1)、PE(X, 2)、PE(X, 4)、PE(X, 6)の5列がそれぞれ稼動となる。休止となったPE列のPE(5, Y)、PE(7, Y)(図中、×

印)は行方向のバイパスを行い、PE(X, 3)、PE(X, 5)、PE(X, 7)はそれぞれ列方向のバイパスを行う。

【0206】特に休止の行と列の交差にあるPE(3, 5)、PE(3, 7)等は縦方向と横方向の双方をバイパスする。2次元格子トラス結合において、仮想プロセッサを実現する際も、1次元リング結合で挙げた例と同様にして行うことが出来る。

【0207】3次元格子トラス結合の場合は、休止の対象が2次元の行／列に対し、面が対象となる。

【0208】上記バイパス回路はPEと同一チップに納めたときに有効となる。すなわち1方向に複数のチップをバイパスする場合、バイパス経路がチップ外にある場合よりもはるかに高速にバイパスを行うことが出来る。特に複数PE/chipの場合、さらにバイパスの高速化を行うことが出来る。

【0209】次にメモリアクセスに関して説明する。例としてまず16台並列の1次元リング結合の並列計算機を挙げる。上記例で16台並列の1次元リング結合型計算機で問題サイズ10の処理を行う場合を挙げたが、この時上記並列計算機が分散メモリ型であり、各PE0～15がローカルメモリ(LM)を持つ場合、休止となっているPEのローカルメモリも使用されないことになる。

【0210】例えば問題サイズは4だが、1つのタスクが必要とするメモリ量が1PEの持つメモリの数倍の容量を必要とする場合もある。通常この場合は1つのタスクのデータを複数に分け、おおもとのデータ保持部から随時転送する方法をとる。しかしこの場合、データを分割したためにデータが分割されていることを意識したプログラムが必要となる。

【0211】そこで、休止しているPEがある場合、同PEのローカルメモリを、稼動しているPEが使用できるようにすることにより、1PEあたりのメモリ容量を増やすことができる。

【0212】図14に16台並列1次元リング結合の並列計算機の例を示す。同例で、上記バイパスに関する実施例を条件とすると、本並列計算機は4台から16台までのスケラブル並列計算機となる。

【0213】今、問題サイズが8とすると、稼動するPEはPE0, 2, 4, 6, 8, 10, 12, 14の8台となる。各PEが自PEの右側のローカルメモリ(LM)44を切り換えスイッチ45で直接アクセス出来るようにすると、8台並列で、各PEのローカルメモリ44の容量が2倍となる。

【0214】問題サイズが5の場合は、稼動PEをPE0, 3, 6, 9, 12とし、PE0はPE1, 2のローカルメモリ44を、PE3はPE4, 5のローカルメモリ44を使用することにより、3倍のメモリ容量となる。4台のPEが稼動の場合は4倍のメモリ容量とな

る。アクセスのためのアドレス空間は、キャッシュメモリで用いるようなバンク方式あるいはウィンドウ方式も考えられるが、連続したアドレス空間として使用しても良い。

【0215】ここで、どのPEを休止とし、どの稼動PEがどの休止PEのローカルメモリ44を使用できる用にするかを決定することが問題となる。まず稼動PEをPE0, 1, 2, 3, 4, 5, 6, 7とし、PE8～15を休止として、PE0がPE8のメモリを、PE7がPE15のメモリを使用することにしても良いが、まず稼動するPEは上記バイパスにおける制約条件もあるが、各PEが番号順に左から順に実装されている場合、PE0からPE8への物理的距離が遠くなる。そのためアクセス対象は実装時のモジュール間の距離を考慮して決定すべきである。

【0216】上記例では、割り付けられるタスクの大きさが均等の場合を念頭に述べたが、タスクの大きさすなわち必要となるメモリ量が異なっても良い。すなわち今6個のタスクがあり、それぞれ必要となるメモリ量が、4, 2, 3, 4, 1, 2であった場合、稼動PEはPE0, 4, 6, 9, 13, 14となり、PE0はPE1, 2, 3をPE4はPE5を、PE6はPE7, 8を、PE9はPE10, 11, 12を、PE14はPE15のローカルメモリ44をそれぞれ使用する。

【0217】このように、単純にタスクを左から割り付けるとすると、タスクの大きさに適応するためには、前述した通信時間の関係から、全てのPEが他の少なくとも3つのPEにアクセスする経路を設けなければならず、回路／配線の増大を招く。これを解消するため、例えばPE0, 4, 8, 12の各PEは自PEの右隣3PEへの経路をもち、他の偶数番号PEは隣のPEへの経路を持つように限定し、大きさの異なるタスクの割り付けはスケジューリングを行い対処する。すなわちPEをグループ化する。

【0218】これは複数PEを1つのチップに載せた場合にも有利となる。例えば、4PEを1つのチップに配置し、上記切り換えスイッチ45などのアクセス回路もチップ内に納める。これにより、アクセス回路の切り換えによる速度低下を防ぐことが出来る。同一チップ上ならば、多少の回路／配線が増加しても速度の低下は防ぐことが出来る。そのため、同一チップに配置したPE内では、全てのPEが他のPEのローカルメモリ44を使用できるようにしても良い。こうすることにより、大きさの異なるタスクへの対処の柔軟性は上がる。

【0219】次に2次元格子トラス結合のメモリアクセスの場合を示す。2次元の場合、休止の対象は行／列である。そのため、アクセス回路も行方向と列方向に設ける。上記1次元リング結合で、チップに載せることも考慮して、PEをグルーピングしたが、2次元の場合でも同様のことがいえる。

【0220】図15に 2×2 のグループを考える。PE(0, 0)は他のPEのローカルメモリ44が使用できる。PE(1, 0)とPE(0, 1)はPE(1, 1)のローカルメモリ44が使用できるようにする。これにより列方向、行方向のスケーリングを利用してメモリ容量の増加を行うことが出来る。また、PE(1, 0)とPE(0, 1)は休止するため、バスパス回路46を設けている。

【0221】3次元格子結合に関しても上記例と同様にすることにより、メモリ容量の増加を行うことが出来る。

【0222】次にハイパーキューブでのメモリアクセスについて説明する。並列計算機の静的結合方式の1つであるハイパーキューブは、通信距離の半径が小さいなどの利点が挙げられるが、スケーラビリティの点でも有利である。すなわちハイパーキューブの次元を1つ下げると、余分な次元に対応する通信経路を使用しなければよく、特別な回路は必要ない。

【0223】例として4次元から3次元にする例を図16に示す。稼動するPEをPE0~7とし、稼動しないPEをPE8~15とすると、容易に1次元下がったハイパーキューブ結合の並列計算機となる。今、PE8~15が休止であるので上記メモリアクセス路の例で述べたように、休止のPEのローカルメモリ44を、稼動するPEのメモリとして使用することが出来る。

【0224】すなわちPE8のローカルメモリ44をPE0が使用し、PE9のローカルメモリ44をPE1が使用する。通信経路0-8, 1-9, 2-10, 3-11, 4-12, 5-13, 6-14, 7-15を使用しないので、メモリアクセス用バスとして上記通信経路を使用しても良い。

【0225】次に複数PEを1つのチップに配置する場合の配線について説明する。図17に 2×2 の4つのPE47a, bを1つのチップ48に載せた場合の例を示す。PE47のモジュールをチップ48内に配置する場合、各PE間の通常の通信路の配線が場所を取る。そこで、アルミ等の配線層を複数化する近年のチップ製造技術を用い、休止PE47aをバイパスする経路を休止PE47aの上を通過するように配線する。これにより、配線領域を縮小することが出来、余分な配線の引き回しによる遅延の増大を防ぐことが出来る。

【0226】同様に、休止PE47aのローカルメモリ44へのアクセス経路もPE47aの上を通過するように配線することにより、配線領域の縮小が行える。

【0227】第4の発明

図18は、第4の発明の並列計算機で用いられる演算プロセッサの実施例である。

【0228】ルータ部51と演算処理部52から構成される。ルータ部51はフリットを蓄えるバッファ53x~53pと、フリットの出力先を決定するスイッチ54

から構成される。

【0229】Pin, Poutはルータ部51と演算処理部52とを接続する通信チャネルである。Xin, Xout, Yin, Youtはそれぞれ隣接する演算プロセッサとの間の入力および出力のための通信チャネルである。上記の演算プロセッサを用いて、例えば図19に示すような並列計算機が構築される。各演算プロセッサ55は、互いに通信チャネル56で接続される。

【0230】図20は本実施例で通信されるメッセージのフォーマットの一例である。一つのメッセージは複数のフリットに分割される。先頭の二つのフリットは図20(a)のフォーマットをしており、図中addressで表されるメッセージの宛先が書かれている。左右方向の宛先が書かれたフリットに上下方向の宛先が書かれたフリットが続く。各方向の宛先は送信側プロセッサと受信側プロセッサとの間の相対距離で表される。

【0231】3つめのフリットからメッセージの本体となる。そのフォーマットは、図20(b)および(c)に示すとおりである。フリットは図中dataで表されるメッセージの格納されている部分と、図中右端のメッセージの継続あるいは終了を表す終了ビット(end_bit)からなる。メッセージが継続する場合は図20(b)のフォーマットのようにent_bitは0であり、メッセージが終了する場合は図3(c)のフォーマットのようにend_bitは1となる。

【0232】図21は本実施例のルータ部51で行われるルーティング方法である。まず左右方向の宛先が一致するまで左右方向に隣接する演算プロセッサ間で通信を行う(ステップ100, 101)。続いて上下方向の宛先が一致するまで上下方向に隣接する演算プロセッサ間で通信を行う(ステップ102, 103)。第4の発明はこのルーティング方法に限定されず、いかなるアルゴリズムにも適応可能である。

【0233】図22はスイッチ54の内部構成例である。図19における左右方向のスイッチ57と上下方向のスイッチ58とからなる。左右方向のスイッチ57と上下方向のスイッチ58は同じ構成で実現される。それを図23に示す。二つの出力先決定回路61と、二入力二出力のクロスバスイッチ62から構成される。クロスバスイッチ62には複数の方向から入力を与えられるので、これらが衝突する場合は調停が行われる。

【0234】図24は出力先決定回路61の構成例である。入力されたフリットのaddress部が宛先比較部63で比較される。内部状態保存部64に記憶されている現在の状態stateで、先頭フリットか否かが判定される。X方向あるいはY方向における宛先が自演算プロセッサと同じメッセージであれば信号eq1を、そうでなければ信号neqを出力先決定部65に出力する。出力先決定部65では、これらの制御信号とフリットのend_bit, そして内部状態stateから出

力先制御信号を発生する。

【0235】`ctl_in`は自プロセッサ（スイッチ58の場合）あるいは次のスイッチ58（スイッチ57の場合）を、`ctl_out`は隣接するプロセッサを出力先を選択するための制御信号である。`ctl_in`、`ctl_out`はいずれもクロスバスイッチ62に入力される。

【0236】出力先決定部65は次状態を決定し、内部状態保存部64に記憶する。さらに出力先決定部65はデクリメンタ66への制御信号`ctl_remove`、`ctl_dec`も発生する。`ctl_remove`はメッセージから先頭のフリットを取り除くための制御信号で、`ctl_dec`は`address`を1減ずるための信号である。

【0237】デクリメンタ66はこれらの制御信号にしたがって、宛先が自演算プロセッサと同じ場合は先頭のフリットを取り除き、そうでないときは先頭のフリットに書かれた`address`を1減ずる。先頭フリットでない場合は、何も操作は行わないでフリットを通過させる。デクリメンタ66はまた、リクエスト信号`req`を発生する。

【0238】図25は内部状態保存部64の状態遷移の様子を表している。初期状態S0にいる時にフリットを受け取った場合には、このフリットは先頭であるので、`address`にしたがってS1あるいはS2に遷移する。`address`が自演算プロセッサと同じ場合にはS1に遷移する。メッセージが続いている場合にはS1あるいはS2の状態を継続し、`end_bit`によりメッセージの終了を検出すると初期状態S0に戻る。

【0239】第4の発明を用いた場合に、従来の問題が解決することを図26を用いて説明する。図26において、55a~55eは演算プロセッサであり、それぞれバッファ53x、53y、53p、スイッチ54および演算処理部52から構成される。

【0240】出力先の演算プロセッサ55bのバッファが一杯で通信が停止しているフリット67が演算プロセッサ55aのバッファ53xに蓄えられているとする。このとき演算プロセッサ55cから別のメッセージのフリット68が入力されるとする。

【0241】このメッセージの出力先が演算プロセッサ55eとすると、演算プロセッサ55eのバッファに空きがあれば、フリット68は出力されるべきである。フリット68はバッファ53yに入力されるので、バッファ53xには関係なく演算プロセッサ55eに出力することができる。

【0242】すなわち、出力先の通信チャンネルが空いてさえいれば、通信の停止している他のメッセージに妨害されることなく、出力可能である。したがって、並列計算機全体としての通信時間が短縮し、スループットも向上する。

【0243】従来例の場合では、先にバッファに蓄えられているフリットの通信が停止してしまうと、それ以降にバッファに蓄えられたフリットは、たとえ出力先のチャンネルが空いていても出力されなかった。しかし第4の発明では、受信されたフリットは、先に出力先を決定しバッファに蓄えられる。したがって、ブロックされている通信チャンネルのメッセージだけが通信を停止し、他のメッセージは通信を妨害されることはない。

【0244】本実施例では二次元のネットワークの場合を説明したが、ルータ部51にスイッチ57を増設するだけで高次元のネットワークに拡張可能である。たとえば三次元の場合は図27のようにスイッチ39を増設すればよい。また通信ネットワークは、本実施例のようなトラス構造に限らず、ハイパーキューブ構造など、他のいかなる構造にも適応可能である。

【0245】さらに第4の発明は、ワームホール・ルーティングに限らず、ストア・アンド・フォワード・ルーティングなど、他のいかなる通信方式にも適応可能である。

【0246】第5の発明

図28は第5の発明の並列計算機で用いられるプロセッサの構成図である。`Xin`、`Xout`はそれぞれ隣接するプロセッサとの間の入力および出力チャンネルである。`Pin`、`Pout`はルータ部71と演算処理部72を接続する通信チャンネルである。ルータ部71は`Xin`、`Pin`から受信したメッセージの転送先を決定し、`Xout`あるいは`Pout`に出力するものである。バイパススイッチ73は、第5の発明のために追加された構成であり、`Xin`からの入力をルータ部71をバイパスして、直接`Xout`に出力するブロックである。

【0247】図29は、図28のプロセッサで構成される並列計算機である。各プロセッサ74a~74eが通信チャンネル75によって接続されている。図示していないが、リクエスト信号とアクノリッジ信号も接続されている。

【0248】本実施例で通信されるメッセージのフォーマットは、図20で示したものと同様である。

【0249】図30はルータ部71の内部構成例である。プロセッサは`Xin`方向のプロセッサからリクエスト信号を受けると、ラッチ76にフリットを受け取る。このフリットがメッセージの先頭であれば、転送先決定ブロック77で自プロセッサに取り込むか、バイパスするか、あるいは隣接するプロセッサに転送するかを決定する。転送先決定ブロック77は同時に出力先へのリクエスト信号を生成する。クロスバスイッチ78にはメッセージとリクエスト信号、そして転送先決定ブロック77からメッセージの出力先を示す信号が与えられる。この信号は2方向から与えられるので、これらが衝突した場合には調停を行って、メッセージとリクエスト信号を出力先へスイッチする。その結果、フリットはバッファ

79に入れられる。同時に転送先決定ブロック77の発したリクエスト信号は出力先のプロセッサに与えられる。

【0250】図31は転送先決定ブロック77を構成するブロック図である。入力されたフリットが、メッセージの先頭である場合は宛先比較ブロック81で宛先を比較する。先頭フリットであるか否かは、状態記憶ブロック82に記憶されている現在の状態 *state* で判定される。宛先はプロセッサ間の相対距離で表されるので、宛先が0ならば自プロセッサ宛、0以外ならば他のプロセッサ宛である。

【0251】自プロセッサ宛ならば信号 *eq1* を、他のプロセッサ宛ならば信号 *neq* を、出力先決定ブロック83に出力する。出力先決定ブロック83はこれらの制御信号と内部状態にしたがって出力先制御信号を発生する。

【0252】*ctl_bypass* はバイパスするための制御信号で、*ctl_out1*、*ctl_out2* はそれぞれ、バイパスしない場合の出力先として *Xout*、*Pout* を選択するための制御信号である。

【0253】*ctl_bypass* は第5の発明において特徴となる制御信号であり、バイパススイッチ73に入力されて、バイパス動作を行う。*ctl_out1*、*ctl_out2* はいずれもクロスバススイッチ78に入力されて、出力先を要求する。また出力先決定ブロック83は次の状態 *next_state* を決定し、状態記憶ブロック82に記憶する。さらに出力先決定ブロック83はデクリメンタ124に対する制御信号 *ctl_remove*、*ctl_dec* も発生する。

【0254】*ctrl_remove* は、デクリメンタ84に入力される宛先を取り除くための制御信号で、*ctl_dec* は宛先を1減ずるため制御信号である。デクリメンタ84はこれらの制御信号にしたがって、宛先が自プロセッサであるときには宛先を取り除き、そうでないときには宛先を1減ずる。先頭フリットでない場合は、何もしないでフリットを通過させる。デクリメンタ84はまたリクエスト信号 *req* を発生する。

【0255】図32は状態記憶ブロック82の状態遷移を表す図である。初期状態 *S0* のときにフリットを受け取ると、このフリットはメッセージの先頭であるので、宛先にしたがって *S1*、*S2* あるいは *S3* に状態を遷移する。例えば、メッセージをバイパスする場合は *S1* に、*Xout* に出力する場合は *S2* に、*Pout* に出力する場合は *S3* に遷移する。

【0256】メッセージが続いている間は *S1* ~ *S3* の状態に留まり、*end_bit* によりメッセージの終了を検出すると初期状態 *S0* に戻る。 *S1* から *S0* に状態が遷移するとバイパスは解除される。すなわちバイパスしている状態でも、メッセージはルータ部71に取り込まれている。ただし、取り込まれるメッセージは終了を

検出する目的だけに用いられ、中継されたりはしない。

【0257】第5の発明のバイパス動作を以下に説明する。図33は図28で示したプロセッサを用いて構成した並列計算機である。85aから85eはメッセージを通信するためのチャネル（図中、*data*）、86aから86eはリクエスト信号、87aから87eはアクノリッジ信号である。本実施例ではメッセージは単方向に通信されるが、双方向通信の場合も本質的な違いはない。各プロセッサは図33(a)に示すように隣接するプロセッサ74とチャネルおよび信号で接続される。

【0258】本実施例で、プロセッサ74aからプロセッサ74eに中継プロセッサを全てバイパスしてメッセージを送信する場合を説明する。プロセッサ74aはプロセッサ74bへリクエスト信号86aを発し、宛先を持ったフリットを74bに送信する。プロセッサ74bはリクエスト信号を受け取るとフリットを受信し、フリットに書かれている宛先を読みだす。

【0259】プロセッサ74bは宛先ではないので、次の転送先プロセッサ74cを選択する。同時にプロセッサ74bは通信チャネル85aと85b、リクエスト信号86aと86b、およびアクノリッジ信号87aと87bをバイパスする。リクエスト信号86aと86bがバイパスされることにより、プロセッサ74cにリクエスト信号が要求されることになる。

【0260】同様な操作がプロセッサ74c及びプロセッサ74dで実行され、図33(b)のように経路が確定される。プロセッサ74eはリクエスト信号を受けると、フリットを受信して宛先を読む。プロセッサ74eは受信側プロセッサであるので、プロセッサ74dに対してアクノリッジ信号を返す。このときプロセッサ74bからプロセッサ74dの内部でアクノリッジ信号87aから87dはバイパスされているので、図33(c)のようにアクノリッジ信号は直ちにプロセッサ74aに到達する。

【0261】こうしてプロセッサ74aとプロセッサ74eとの間のハンドシェイクが完成し、メッセージを転送する経路が確立される。以後は図33(d)のようにプロセッサ74aとプロセッサ74eの間で2つめ以降のフリットの通信が行われる。この間プロセッサ74b~74dは、終了検出を行うためにバイパスしているフリットを監視しつつける。そしてメッセージの終了を検出すると初期状態に戻る。

【0262】この方法を用いた場合に転送に必要な時間は、経路を確立するためのリクエスト信号の伝達に *D*、アクノリッジ信号の伝達に1、経路確立後のメッセージの送信に $2 \times (L - 1)$ 必要で、全体で $(D + 2 \times L - 1)$ となる。このようにプロセッサ間の通信時間が短縮される。*D* = 4、*L* = 4 の場合のタイミング図を図34に示す。

【0263】以上はプロセッサ間でのメッセージの通信

をハンドシェイクで行った場合であるが、さらに通信を高速にするために、図33(b)、(c)の処理によって経路を確立させたのち、図33(e)のようにプロセッサ74aとプロセッサ74eをクロック同期させて通信を行う方法が考えられる。この場合、メッセージの送信に必要な時間は $(L-1)$ となっており、全体では $(D+L)$ となる。これは従来のワームホール・ルーティングのほぼ $1/2$ の時間で、プロセッサ間の通信にかかる時間は大いに短縮される。 $D=4$ 、 $L=4$ の場合のタイミング図を図35に示す。

【0264】同期通信の場合は、プロセッサ間の距離が非常に大きいと、1クロックではフリットが到達できない可能性がある。このような場合を考慮して、通信を行っているプロセッサの間に存在するプロセッサの一部でメッセージを中継することを考える。すなわち、上述した実施例では、自プロセッサ宛でないメッセージは全てバイパスしていたが、転送先への距離に応じてバイパスと中継を分けるように、宛先比較ブロック81および出力先決定クロック83を変更する。

【0265】本実施例を用いて、例えば1クロックで到達可能な距離を8とした場合を考える。先頭のフリットに書かれているデータは、受信側プロセッサまでの相対距離であるので、この宛先が8の倍数であるときには中継し、そうでないときにはバイパスするように、宛先比較ブロック81と出力先決定ブロック83を変更する。

【0266】こうして1クロックで到達できる距離毎に中継プロセッサが選択される。このような変更で、非常に規模の大きなネットワークでも、従来例よりも高速にプロセッサ間通信を行うことができる。

【0267】なお、本実施例では一次元の場合を説明したが、第5の発明は一次元に限らず、容易に多次元に拡張可能である。図36は二次元の場合のプロセッサ構成図である。 X_{in} 、 Y_{in} 、 P_{in} からの入力を、 X_{out} 、 Y_{out} 、 P_{out} に出力する。バイパススイッチ88はバイパススイッチ73に比べてバイパスする信号が一つ増えたブロックである。

【0268】このように次元を拡張するには、次元数 N 分だけルータ部71とバイパススイッチ73を用意する。これで各次元でバイパスが可能になるが、異なる次元間でもバイパス可能にするためには、バイパススイッチ88のように他の次元からのバイパスも選択できるように、スイッチの入力を増やす。

【0269】

【発明の効果】以上のように、第1の発明の並列計算機は、並列度がプロセッサの数よりも大きい場合は通常のSIMD型並列計算機と同様に実行することができ、並列度がプロセッサの数よりも少ない場合には、複数命令の並列実行を行なうことにより、実行時間の短縮とプロセッサの有効利用を図ることができる。また、通常のSIMD型並列計算機とオブジェクト互換をもつようにす

ることが可能である。

【0270】また、第2の発明の並列計算機によれば、SIMD型並列計算機にわずかなハードウェアを付加するだけで、MIMD処理が可能になり、より柔軟性の高い処理が可能になる。

【0271】第3の発明の並列計算機によれば、処理する問題サイズに応じて使用しないPEをバイパスすることにより、結合形態を維持したまま、問題サイズに最適な構成を取ることが出来る。使用しないPEのメモリを、使用するPEがアクセスできるようにすることにより、各PEのメモリ容量を増やすことが出来る。

【0272】さらに、同一チップ内において、バイパス経路、メモリアクセス経路をPEの上を通過するように配線することにより、配線領域の縮小、遅延増大の防止を行うことが出来る。

【0273】第4の発明の並列計算機によれば、フリットは出力先毎に出力側のバッファに蓄えられるので、通信可能なメッセージは、通信が停止している他のメッセージによって妨害されることはない。その結果、通信時間が短縮され、スループットが向上する。

【0274】さらに、第5の発明の並列計算機によれば、中継プロセッサをバイパスすることにより、通信に必要な時間を大いに短縮することができる。このため、特にネットワークの規模が非常に大きくプロセッサ間の距離が大きな場合に有効である。

【図面の簡単な説明】

【図1】第1の発明における並列計算機の一実施例の構成図。

【図2】第1の発明における実施例で用いた処理の一例。

【図3】並列度が大きい場合の図2の処理を行なうために供給される命令列。

【図4】並列度が小さい場合の図2の処理を行なうために供給される命令列。

【図5】第1の発明の図1と異なる実施例の構成図。

【図6】第2の発明の並列計算機に係わる一実施例の構成を示すブロック図。

【図7】第2の発明の中心となる演算プロセッサの構成例を示す図。

【図8】図7で示したローカルメモリ中に作成されるアドレステーブルのメモリマップ図。

【図9】図7と異なる演算プロセッサの構成例を示す図。

【図10】第3の発明における1次元リング結合並列計算機のバイパスに関する一例の構成図。

【図11】第3の発明においてバイパス回路として用いた双方向バスの詳解図。

【図12】第3の発明においてバイパス回路として用いたハンドシェイク用信号線の詳解図。

【図13】第3の発明における2次元トーラス結合並列

計算機のバイパスに関する一例の構成図。

【図 14】第 3 の発明における 1 次元リング結合並列計算機のメモリアクセスに関する一例の構成図。

【図 15】第 3 の発明における 2 次元トーラス結合並列計算機のバイパス及びメモリアクセスに関する一例の構成図。

【図 16】第 3 の発明におけるハイパーキューブ結合並列計算機のメモリアクセスに関する一例の構成図。

【図 17】第 3 の発明におけるバイパス経路及びアクセス経路のチップ上での配線図。

【図 18】第 4 の発明における演算プロセッサの一実施例を表す構成図。

【図 19】図 18 で示した演算プロセッサで構成される並列計算機の構成図。

【図 20】第 4 の発明において通信されるメッセージのフォーマット例。

【図 21】第 4 の発明におけるルーティング方法を表すフローチャート。

【図 22】図 18 で示したスイッチの内部構成図。

【図 23】図 22 で示した一方向のスイッチの内部構成図。

【図 24】図 23 で示した出力先決定回路の内部構成図。

【図 25】図 23 で示した出力先決定回路の状態遷移図。

【図 26】図 18 で示した演算プロセッサを用いた通信方法を表す図。

【図 27】第 4 の発明における 3 次元ネットワークの場合の一実施例を示す構成図。

【図 28】第 5 の発明におけるプロセッサの一実施例の構成を示す構成図。

【図 29】図 28 で示したプロセッサを用いた並列計算機の構成図。

【図 30】図 28 で示したルータ部の一例を示す図。

【図 31】図 30 で示した転送先決定ブロックの構成図。

【図 32】図 30 で示した転送先決定ブロックの状態遷移図。

【図 33】第 5 の発明におけるバイパス動作を表す図。

【図 34】第 5 の発明におけるバイパス動作時の転送時間を表すタイミング図。

【図 35】第 5 の発明におけるクロック同期による転送時間を表すタイミング図。

【図 36】第 5 の発明における 2 次元の場合のプロセッサ構成図。

【図 37】第 2 の発明に対する従来の演算プロセッサの構成を示す図。

【図 38】第 4 の発明に対する従来の演算プロセッサの構成図。

【図 39】図 38 で示した演算プロセッサを用いた従来

の通信方法を表す図。

【図 40】第 5 の発明に対する従来のプロセッサ構成図。

【図 41】第 5 の発明に対する従来のバイパス動作時の転送時間を表すタイミング図。

【図 42】第 5 の発明に対する従来のクロック同期による転送時間を表すタイミング図。

【符号の説明】

- 1 プロセッサ
- 2 プロセッサアレイ制御装置
- 3 セレクタ

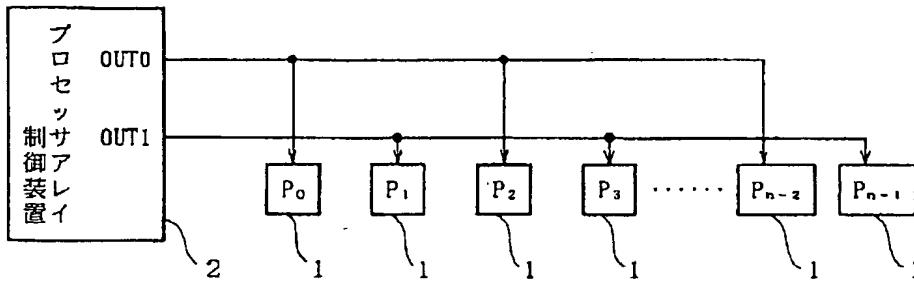
OUTO, 1 出力端子

- 11 制御プロセッサ
- 12 演算プロセッサ
- 13 全体メモリ
- 14 ローカルメモリ
- 15 グローバルバス
- 16 ネットワーク
- 17 同期バス
- 18 フロントエンド計算機
- 21 メモリアドレスバス
- 22 メモリデータバス
- 23 マルチプレクサ
- 24 命令レジスタ
- 25 プログラムカウンタ
- 26 アドレスレジスタ
- 27 データレジスタ
- 28 内部バス
- 29 ALU
- 30 レジスタファイル
- 31 状態制御レジスタ
- 32 プロセッサ番号レジスタ
- 33 通信制御部
- 41 双方向バッファ
- 42 識別回路
- 43 セレクタ
- 44 ローカルメモリ
- 45 切り換えスイッチ
- 46 バスパス回路
- 47 PE (演算要素)
- 48 チップ
- 51 実施例のルータ部
- 52 演算処理部
- 53 x, 53 y, 53 p バッファ
- 54 スイッチ
- 55 a ~ 55 e 演算プロセッサ
- 56 通信チャネル
- 57, 58, 69 一方向のスイッチ
- 61 出力先決定回路
- 62 クロスバススイッチ

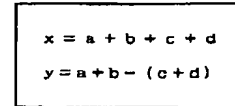
63 出力先選択部
 64 内部状態保存部
 65 出力先決定部
 66 デクリメンタ
 71 ルータ部
 72 演算処理部
 73, 88 バイパススイッチ
 74a~74e プロセッサ
 75, 85a~85e 通信チャンネル
 76 ラッチ

77 転送先決定ブロック
 78 クロスバスイッチ
 79 バッファ
 81 宛先比較ブロック
 82 状態記憶ブロック
 83 出力先決定ブロック
 84 デクリメンタ
 86a~86e リクエスト信号
 87a~87e アクノリッジ信号

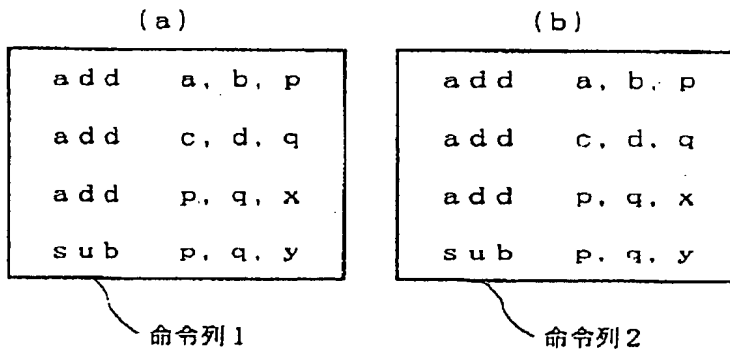
【図1】



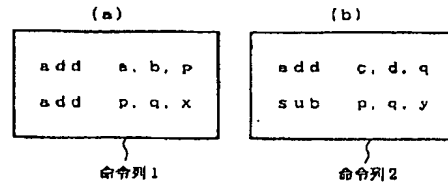
【図2】



【図3】

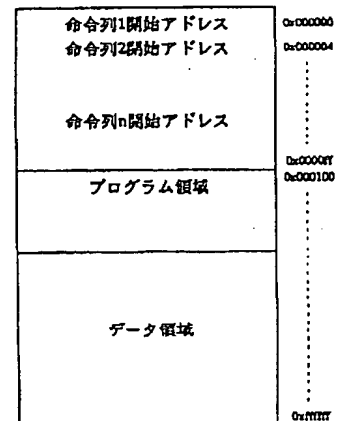
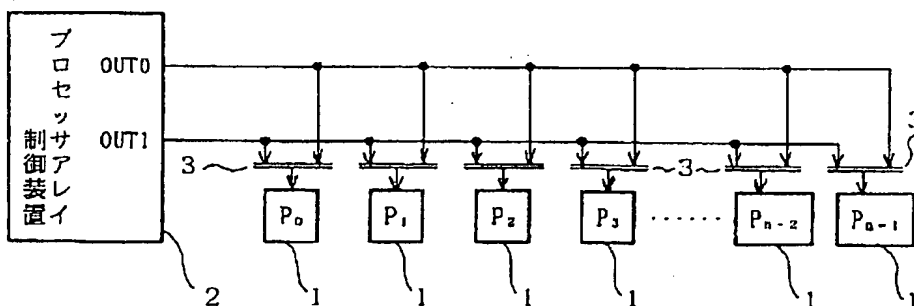


【図4】

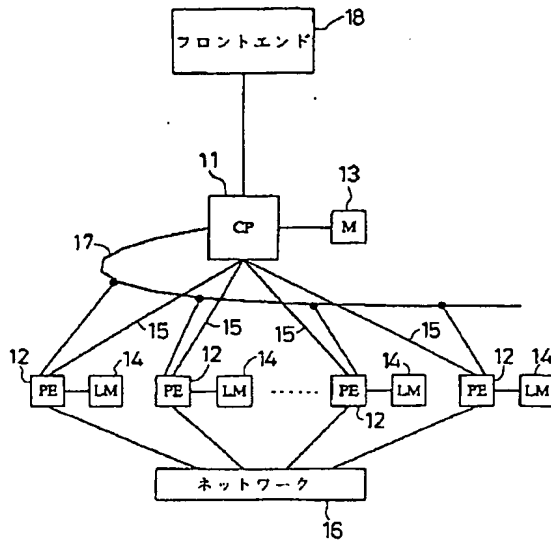


【図8】

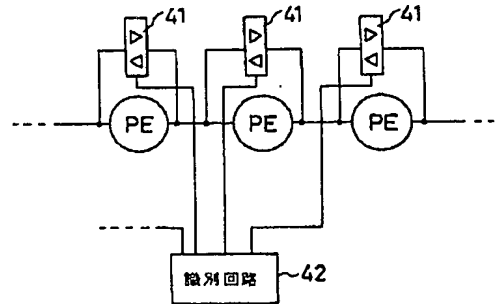
【図5】



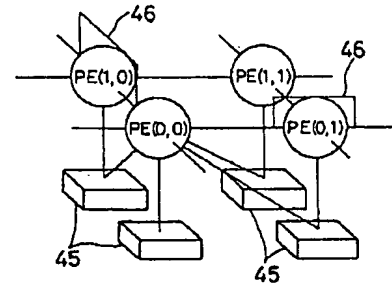
【図 6】



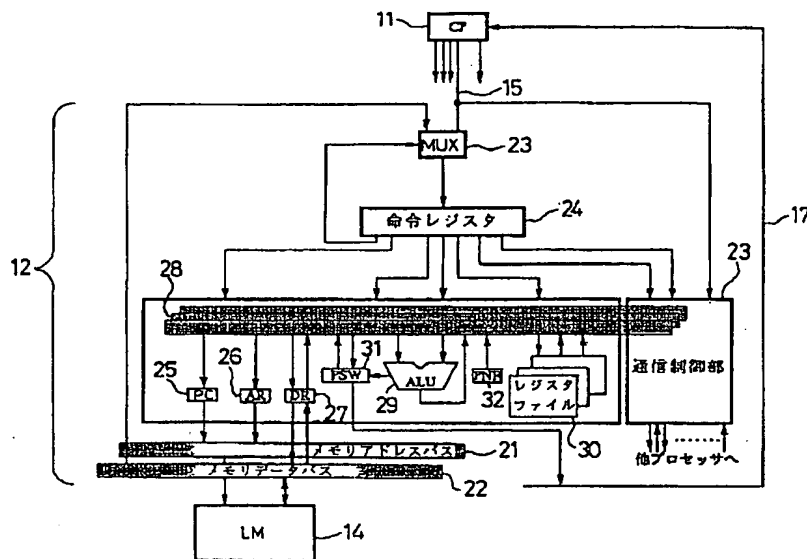
【図 11】



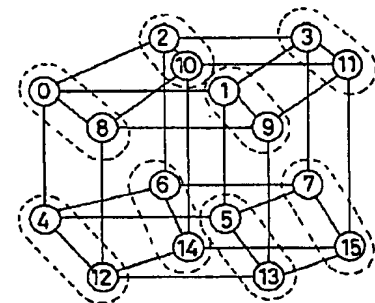
【図 15】



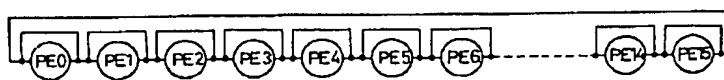
【図 7】



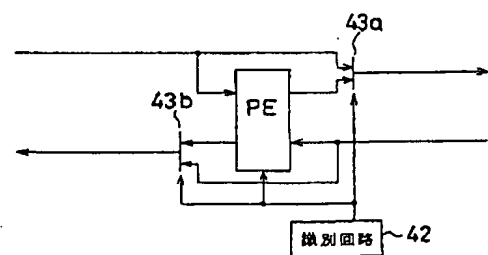
【図 16】



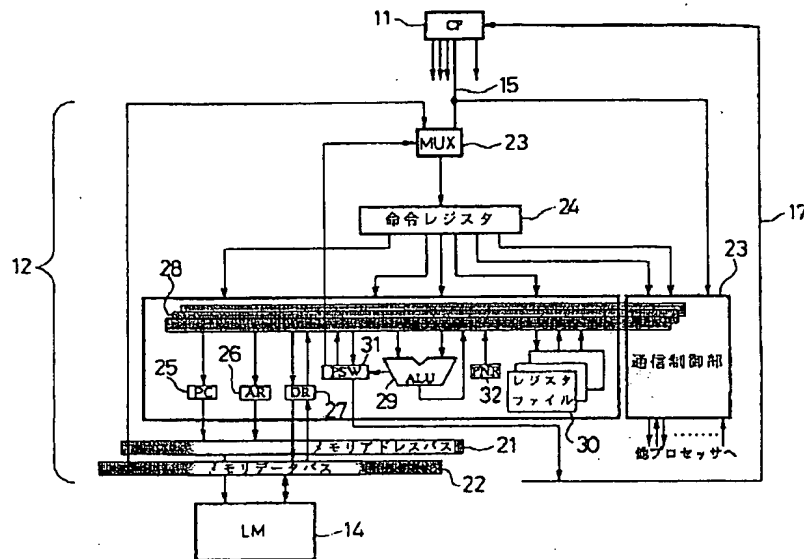
【図 10】



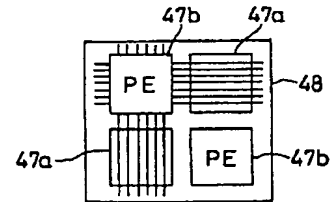
【図 12】



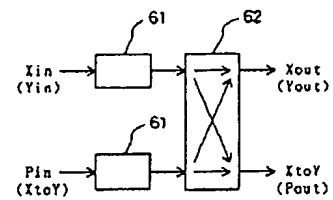
【図 9】



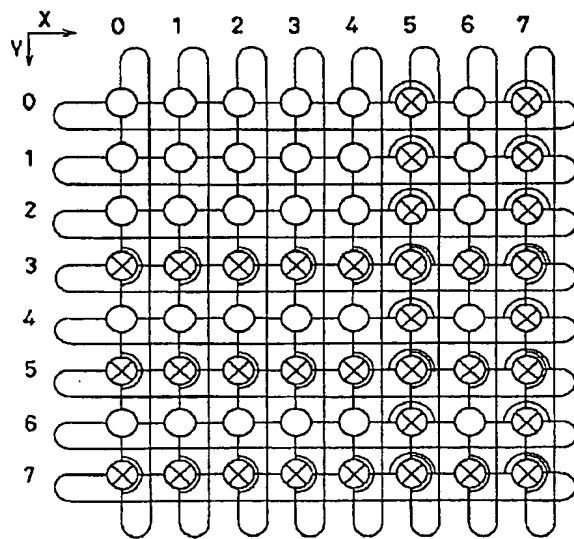
【図 17】



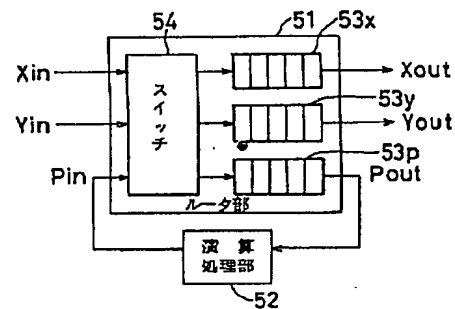
【図 23】



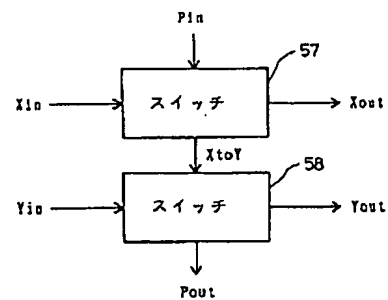
【図 13】



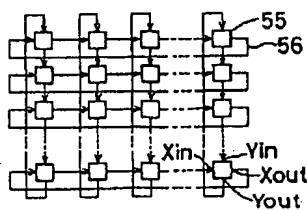
【図 18】



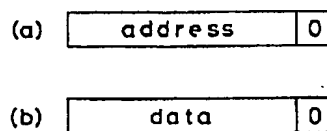
【図 22】



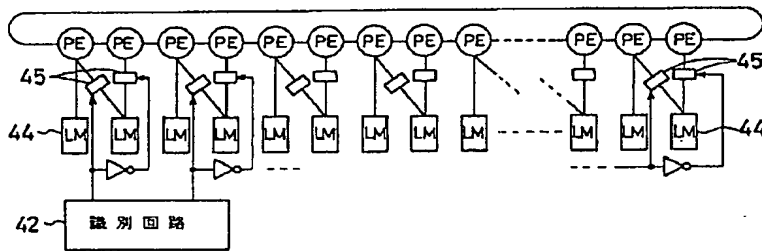
【図 19】



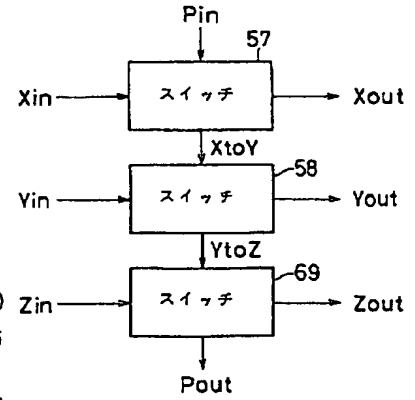
【図 20】



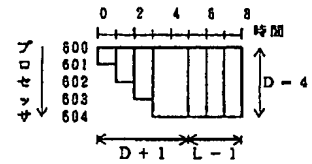
【図 14】



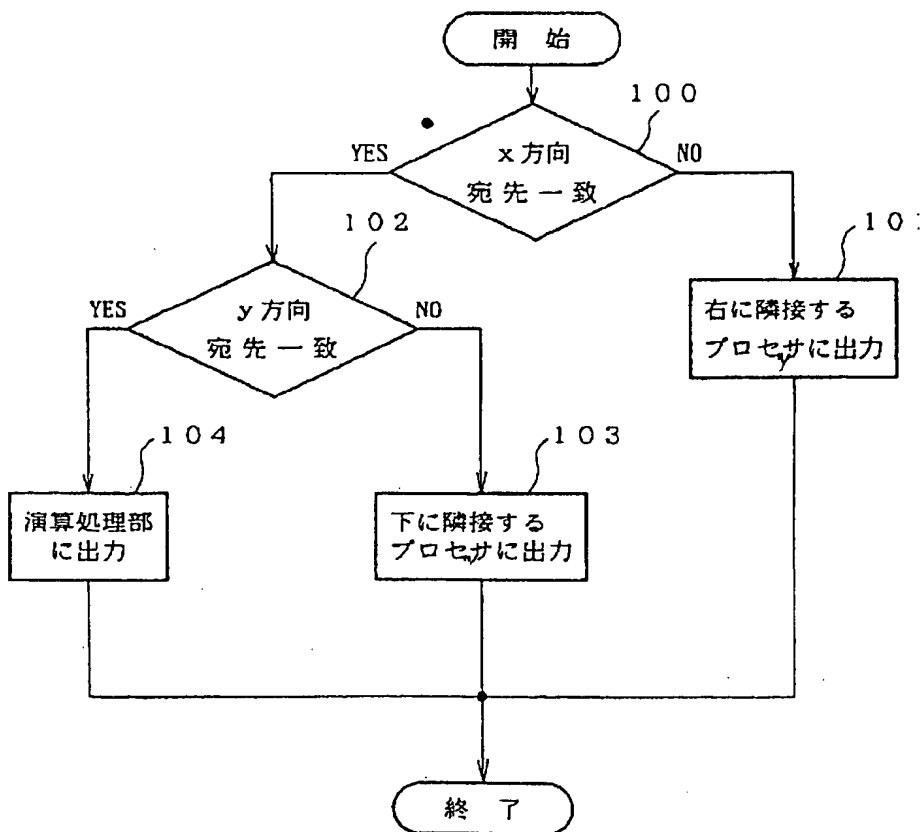
【図 27】



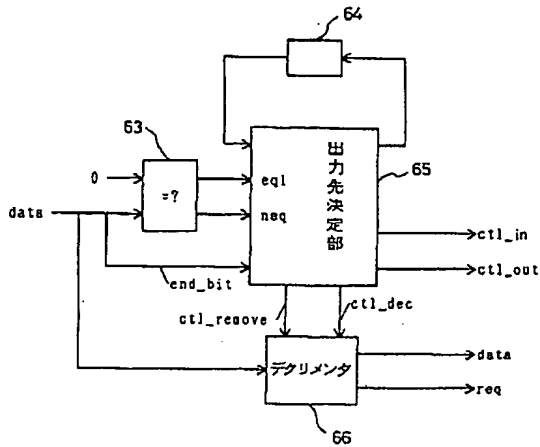
【図 35】



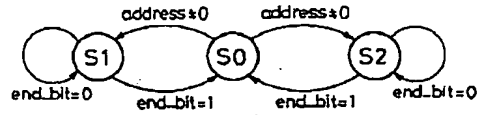
【図 21】



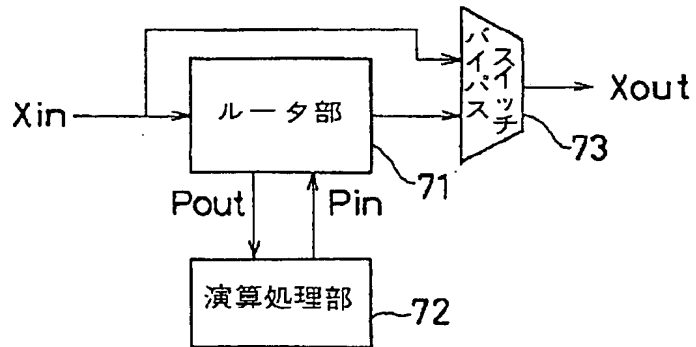
【図24】



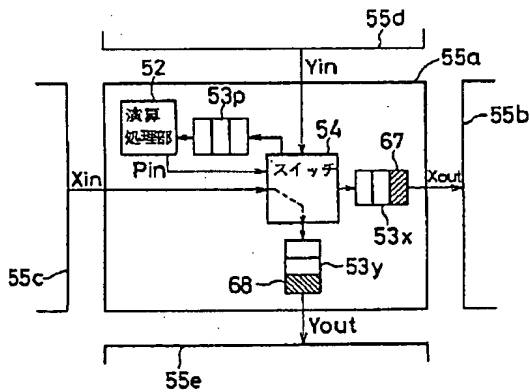
【図25】



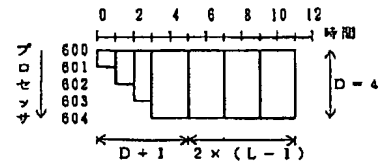
【図28】



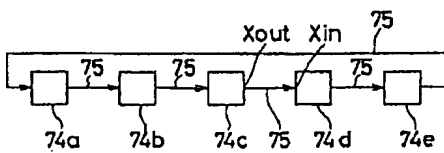
【図26】



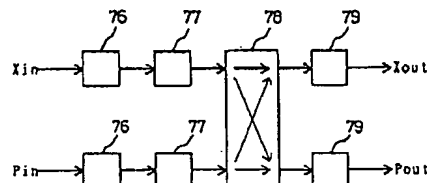
【図34】



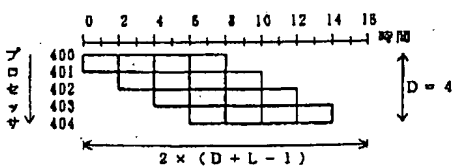
【図29】



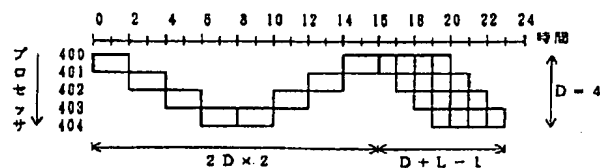
【図30】



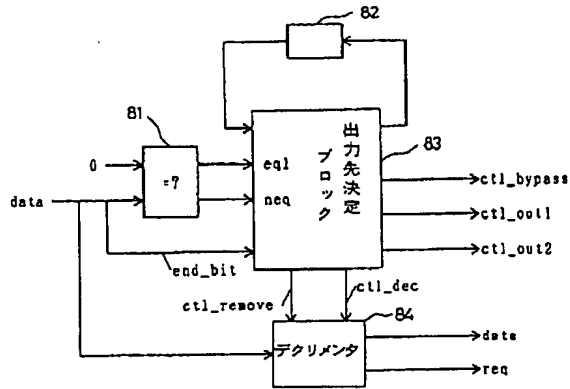
【図41】



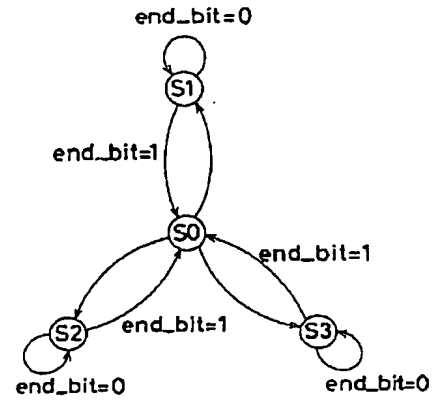
【図42】



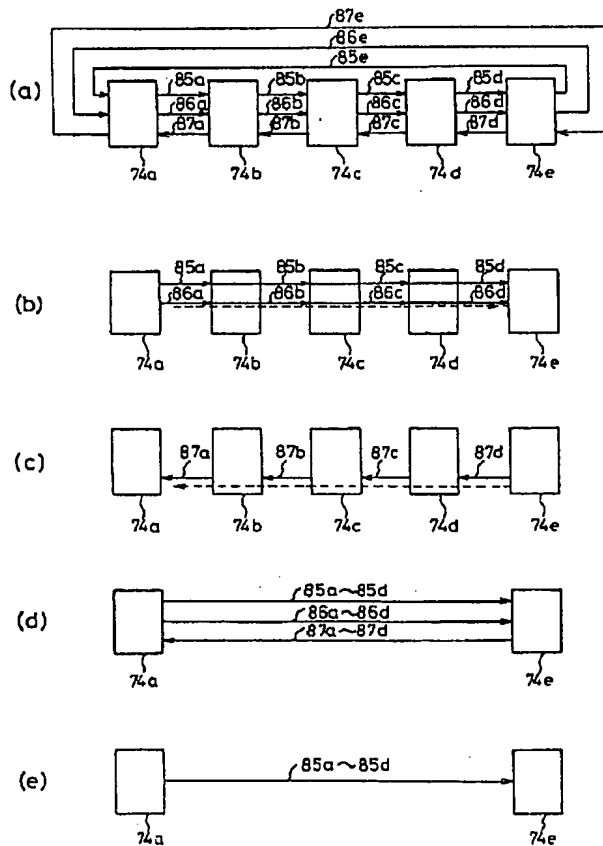
【図 3 1】



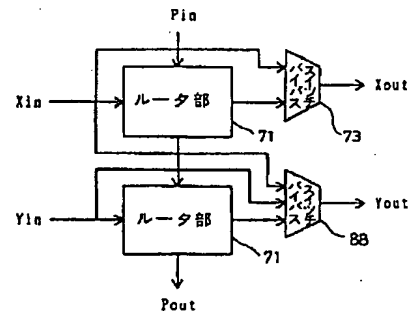
【図 3 2】



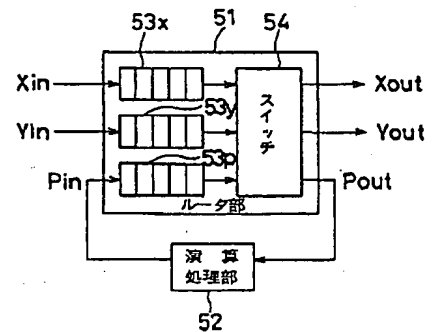
【図 3 3】



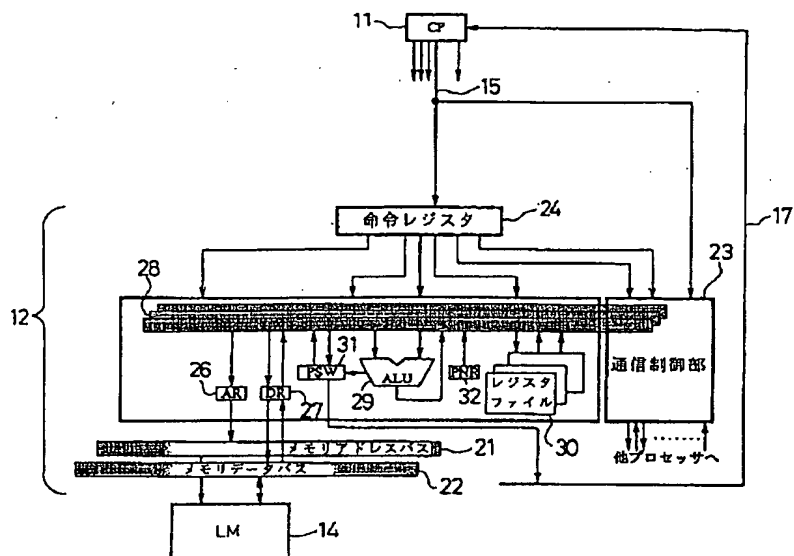
【図 3 6】



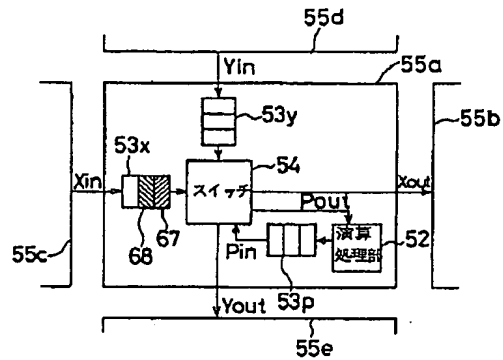
【図 3 8】



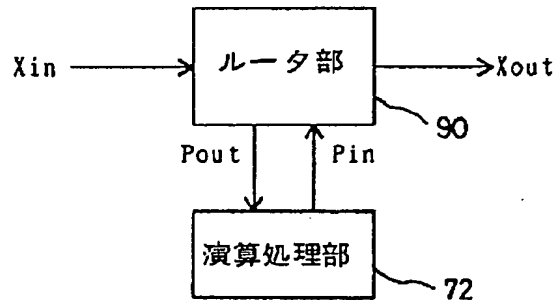
【図 37】



【図 39】



【図 40】



フロントページの続き

(72) 発明者 高橋 真史
 神奈川県川崎市幸区小向東芝町 1 株式会
 社東芝総合研究所内